UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de
Ingeniería y Sistemas de Telecomunicación

PROYECTO FIN DE GRADO

Análisis de datos y modelado simbólico para aplicación de detección temprana de trastornos del lenguaje

Óscar Palomo Díaz

Grado en Ingeniería Telemática

Septiembre de 2014

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Análisis de datos y modelado simbólico para aplicación de detección temprana de trastornos del lenguaje

AUTOR: Óscar Palomo Díaz

TITULACIÓN: Grado en Ingeniería Telemática

TUTORA: Mª Luisa Martín Ruiz

DEPARTAMENTO: Departamento de Ingeniería y Arquitecturas Telemáticas

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Jorge Bonache Pérez

TUTORA: Mª Luisa Martín Ruiz

SECRETARIO: Javier Malagón Hernández

Fecha de lectura: 15 de septiembre de 2014

Calificación:

El Secretario,

A mi tutora, Marisa, por la confianza depositada en mí y su dedicación constante.

A mis padres y mi hermana, por su apoyo, paciencia y amor.

Y especialmente, a mi tía Paqui y mi tío Pedro, por tener el corazón tan grande; sin ellos, nada de esto hubiera sido posible.

Resumen

Este Proyecto Fin de Grado trabaja en pos de la mejora y ampliación de los sistemas Pegaso y Gades, dos Sistemas Expertos enmarcados en el ámbito de la e-Salud. Estos sistemas, que ya estaban en funcionamiento antes del comienzo de este trabajo, apoyan la toma de decisiones en Atención Primaria. Esto es, permiten evaluar el nivel de adquisición del lenguaje en niños de 0 a 6 años a través de sus respectivas aplicaciones web. Además, permiten almacenar dichas evaluaciones y consultarlas posteriormente, junto con las decisiones del sistema asociadas a las mismas. Pegaso y Gades siguen una arquitectura de tres capas y están desarrollados usando fundamentalmente componentes Java y siguiendo.

Como parte de este trabajo, en primer lugar se solucionan algunos problemas en el comportamiento de ambos sistemas, como su incompatibilidad con Java SE 7. A continuación, se desarrolla una aplicación que permite generar una ontología en lenguaje OWL desde código Java. Para ello, se estudia primero el concepto de ontología, el lenguaje OWL y las diferentes librerías Java existentes para generar ontologías OWL.

Por otra parte, se mejoran algunas de las funcionalidades de los sistemas de partida y se desarrolla una nueva funcionalidad para la explotación de los datos almacenados en las bases de datos de ambos sistemas Esta nueva funcionalidad consiste en un módulo responsable de la generación de estadísticas a partir de los datos de las evaluaciones del lenguaje que hayan sido realizadas y, por tanto, almacenadas en las bases de datos. Estas estadísticas, que pueden ser consultadas por todos los usuarios de Pegaso y Gades, permiten establecer correlaciones entre los diversos conjuntos de datos de las evaluaciones del lenguaje. Por último, las estadísticas son mostradas por pantalla en forma de varios tipos de gráficas y tablas, de modo que los usuarios expertos puedan analizar la información contenida en ellas.

Abstract

This Bachelor's Thesis works towards improving and expanding the systems Pegaso and Gades, which are two Expert Systems that belong to the e-Health field. These systems, which were already operational before starting this work, support the decision-making process in Primary Care. That is, they allow to evaluate the language acquisition level in children from 0 to 6 years old. They also allow to store these evaluations and consult them afterwards, together with the decisions associated to each of them. Pegaso and Gades follow a three-tier architecture and are developed using mainly Java components.

As part of this work, some of the behavioural problems of both systems are fixed, such as their incompatibility with Java SE 7. Next, an application that allows to generate an OWL ontology from Java code is developed. In order to do that, the concept of ontology, the OWL language and the different existing Java libraries to generate OWL ontologies are studied.

On the other hand, some of the functionalities of the initial systems are improved and a new functionality to utilise the data stored in the databases of both systems is developed. This new functionality consists of a module responsible for the generation of statistics from the data of the language evaluations that have been performed and, thus, stored in the databases. These statistics, which can be consulted by all users of Pegaso and Gades, allow to establish correlations between the diverse set of data from the language evaluations. Finally, the statistics are presented to the user on the screen in the shape of various types of charts and tables, so that the expert users can analyse the information contained in them.

Índice de contenidos

Índice de figuras

Índice de tablas

Lista de acrónimos

Acrónimo	Significado
API	*Application Programming Interface* (Interfaz de Programación de Aplicaciones)
BC	Base de Conocimiento
BD	Base de Datos
CSS	*Cascading Style Sheets* (Hojas de Estilo en Cascada)
DAO	*Data Access Object* (Objeto de Acceso a Datos)
DL	*Description Logic* (Lógica Descriptiva)
EIS	*Enterprise Information System* (Sistema de Información Empresarial)
EJB	*Enterprise JavaBeans*
HTML	*HyperText Markup Language* (Lenguaje de Marcas de Hipertexto)
HTTP	*HyperText Transfer Protocol* (Protocolo de Transferencia de Hipertexto)
IA	Inteligencia Artificial
IC	Ingeniería del Conocimiento
Java EE	*Java Platform, Enterprise Edition* (Plataforma Java, Edición Empresarial)
Java SE	*Java Platform, Standard Edition* (Plataforma Java, Edición Estándar)
JDK	*Java SE Development Kit (*Kit de Desarrollo de Java SE)
JPEG/JPG	*Joint Photographic Experts Group* (Grupo Conjunto de Expertos en Fotografía)
JRE	*Java SE Runtime Environment* (Entorno de tiempo de ejecución Java)
JSP	*JavaServer Pages* (Páginas *JavaServer*)
JVM	*Java Virtual Machine* (Máquina Virtual de Java)
KDD	*Knowledge Discovery in Databases* (Descubrimiento de Conocimiento en Bases de Datos)
OWL	*Web Ontology Language* (Lenguaje de Ontologías Web)
PNG	*Portable Network Graphics*
RDF	*Resource Description Framework* (Marco de Descripción de Recursos)
SBC	Sistema Basado en Conocimiento
SE	Sistema Experto
SQL	*Structured Query Language* (Lenguaje de Consulta Estructurado)
UML	*Unified Modeling Language* (Lenguaje de Modelado Unificado)
URI	*Uniform Resource Identifier* (Identificador de Recursos Uniforme)
URL	*Uniform Resource Locator* (Localizador de Recursos Uniforme)
W3C	*World Wide Web Consortium* (Consorcio *World Wide Web*)
XML	*eXtensible Markup Language* (Lenguaje de Marcas Extensible)

1. Introducción

1.1. Motivación

El presente Proyecto Fin de Grado (PFG) supone una continuación del proceso de desarrollo de las plataformas web de e-Salud Pegaso y Gades. Dichas plataformas han sido caracterizadas, modeladas y descritas por la tutora del presente proyecto, Mª Luisa Marín Ruiz, como parte de su tesis doctoral en curso.

Cada una de las plataformas de e-Salud constituye un servicio telemático cuya finalidad es tratar de facilitar la detección precoz de trastornos del lenguaje en niños de 0 a 6 años:

- **Pegaso** es un sistema que pretende ayudar al pediatra de Atención Primaria (AP) a decidir si derivar o no a atención especializada a un niño con posibles trastornos del lenguaje [MAR11].
- **Gades** es un sistema que pretende facilitar a los profesionales que trabajan en escuelas infantiles (educadores, logopedas, etc.) la identificación precoz de niños con posibles trastornos del lenguaje [MAR13].

Gades y Pegaso tienen como núcleo una Base de Conocimiento (BC) que aglutina el conocimiento y la experiencia de diversos tipos de profesionales con objeto de permitir la identificación de indicadores tempranos de los trastornos del lenguaje. Empleando dicha BC se podrá abordar, lo antes posible, la intervención ante posibles trastornos del lenguaje, consiguiendo derivar a una exploración más específica que ayude a resolver o minimizar las consecuencias del posible trastorno en el desarrollo integral del niño [MAR14].

Ambos sistemas valorarán, a partir de las respuestas introducidas por los usuarios y haciendo uso de su BC, si el nivel de adquisición del lenguaje del niño es normal o, por el contrario, inferirá qué acciones debe recomendar el profesional (pediatra, educador, etc.) para el adecuado tratamiento de un posible trastorno [MAR13].

La diferencia primordial entre Pegaso y Gades radica en los ámbitos a los que están dirigidas, y en la especialización del conocimiento que se recoge en la BC. Concretamente, existe una mayor especialización en la BC de Gades que en la de Pegaso. Esto se debe a que en las escuelas infantiles, ámbito de aplicación de Gades, el tiempo del que los profesionales disponen para realizar una evaluación y un seguimiento continuo del lenguaje del niño es mayor que el de los profesionales que trabajan en los centros de AP, ámbito de aplicación de Pegaso.

1.2. Antecedentes

Este Proyecto Fin de Grado tiene como principales antecedentes los siguientes trabajos:

- El trabajo de investigación titulado "Modelo de conocimiento para detección precoz de trastornos del lenguaje en niños de 0 a 6 años" [MAR11].
- El Proyecto Fin de Grado titulado "Implementación de una arquitectura de componentes para un sistema de apoyo a la toma de decisiones en Atención Primaria" [URI13].

1.3. Objetivos

1.3.1. Objetivo general

El objetivo general de este proyecto es llevar a cabo varias mejoras sobre las funcionalidades ya desarrolladas en Pegaso y Gades y, además, dotar a ambos sistemas de una nueva funcionalidad que permita explotar los datos que almacenan.

1.3.2. Objetivos específicos

De modo más específico, los objetivos de este proyecto son los descritos a continuación:

- A pesar de que Pegaso y Gades funcionan correctamente con el JDK 6 (*Java SE Development Kit* 6), existe un *bug*[1] que causa la disfunción de ambos sistemas al usar el JDK 7, versión posterior al JDK 6. Se desea resolver este *bug*, de modo que Pegaso y Gades funcionen correctamente con ambas versiones.
- Estudiar las diferentes posibilidades para automatizar el proceso de generación de una ontología en lenguaje OWL DL desde código Java y, tras elegir una de ellas, desarrollar una primera versión de una aplicación[2] capaz de realizar dicha tarea. Para ello, será necesario analizar el concepto de ontología y el Lenguaje de Ontologías Web, *Web Ontology Language* (OWL).
- Optimizar algunas de las funcionalidades ya desarrolladas para las plataformas Pegaso y Gades.

[1] Un ***bug*** es un error o fallo que se produce en un programa informático o sistema de software que desencadena un resultado indeseado.

[2] Una **aplicación** es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos [WIK1].

- Analizar, diseñar e implementar, tanto en Pegaso como en Gades, el módulo encargado de la consulta de estadísticas a partir de las evaluaciones del nivel de adquisición del lenguaje de los niños que estén almacenadas en la Base de Datos (BD).
- Desarrollar un manual de usuario para el proceso de consulta de estadísticas de las plataformas Pegaso y Gades.

1.4. Estructura

La presente memoria describe el trabajo realizado en el curso de este Proyecto Fin de Grado y está organizada en varios capítulos cuyos contenidos se detallan a continuación:

- **Capítulo 1: Introducción.** En él se explica la motivación y los objetivos de este Proyecto Fin de Grado, así como la estructura del mismo.
- **Capítulo 2: Marco Tecnológico.** En este capítulo se expone una visión general de los fundamentos tecnológicos de los campos en los que se encuadra este trabajo, así como una descripción detallada del estado y las tecnologías usadas en los sistemas de partida.
- **Capítulo 3: Descripción de las soluciones propuestas.** En él se describen las soluciones propuestas a los diferentes problemas planteados en el proyecto. Se comienza con una descripción general del problema o la necesidad existente, para después dar paso a la descripción de las posibles soluciones, si hay varias, y de la solución escogida en cada caso. En cada solución se justifican las decisiones tomadas.
- **Capítulo 4: Pruebas y resultados.** En este capítulo se pormenorizan todas las pruebas a las que han sido sometidas las soluciones propuestas en el capítulo anterior para comprobar su correcto funcionamiento. Asimismo, se incluyen los resultados obtenidos en dichas pruebas, así como el análisis de los mismos.
- **Capítulo 5: Conclusiones.** En este capítulo se expone una visión general del trabajo realizado y expuesto en los capítulos anteriores, y se plantean las conclusiones derivadas del mismo.
- **Capítulo 6: Trabajo futuro.** En él se proponen una serie de líneas de trabajo futuras que, por estar fuera del alcance de este proyecto, no han sido abordadas.
- **Capítulo 7: Referencias.** En este capítulo se encuentran las fuentes bibliográficas consultadas durante el desarrollo del presente trabajo.

- **Anexo I: Aplicación Java para generar una ontología OWL.** En este anexo se incluye el código de la aplicación desarrollada como parte de este PFG para generar una ontología OWL desde código Java. También se incluye la ontología generada por dicha aplicación.
- **Anexo II: Manual de usuario de la funcionalidad de consulta de estadísticas en Gades.** En este anexo se puede encontrar un manual en el que se explica cómo usar la funcionalidad de consulta de estadísticas, desarrollada como parte de este de PFG, en la plataforma Gades.

2. Marco Tecnológico

En este capítulo, se explican en primer lugar las bases tecnológicas de los campos en los que se encuadran los sistemas Pegaso y Gades (apartado 2.1), y a continuación se detalla el funcionamiento de cada sistema y las tecnologías usadas en ambos (apartado 2.2).

2.1. Bases tecnológicas

Los conceptos necesarios para entender las bases tecnológicas sobre las que se asientan los sistemas Pegaso y Gades son los siguientes:

- El concepto de Sistemas Expertos, que se enmarcan en la disciplina conocida como Ingeniería del Conocimiento (IC).
- El concepto de aplicaciones multicapa.

2.1.1. Ingeniería del Conocimiento: Sistemas Expertos

Antes de introducir el concepto de IC, necesitamos conocer qué es la Inteligencia Artificial (IA). La **Inteligencia Artificial** es un área de la informática que trata de emular, en sistemas artificiales, algunas de las facultades intelectuales humanas. Con inteligencia humana nos referimos típicamente a procesos de percepción sensorial (visión, audición, etc.) y a sus consiguientes procesos de reconocimiento de patrones [BEN13].

En [ALO04] se define la **Ingeniería del Conocimiento** como "la disciplina tecnológica que se centra en la aplicación de una aproximación sistemática, disciplinada y cuantificable al desarrollo, funcionamiento y mantenimiento de Sistemas Basados en Conocimiento". En otras palabras, "desde el punto de vista metodológico, la Ingeniería del Conocimiento ofrece un marco conceptual sólido, basado en las técnicas de la Inteligencia Artificial, que permite desarrollar y validar Sistemas Basados en Conocimiento" [MAR11].

Un **Sistema Basado en Conocimiento** (SBC) se puede definir como "un sistema software capaz de soportar la representación explícita del conocimiento de un dominio específico y de explotarlo a través de los mecanismos de razonamiento para proporcionar un comportamiento de nivel alto en la resolución de problemas" [GUI94].

Ligado al concepto de SBC, se encuentra el de **Sistema Experto** (SE), que es una aplicación diseñada para actuar como un especialista humano en un dominio particular o área de conocimiento. En este sentido, puede considerarse como intermediario entre el experto humano, que transmite su conocimiento al sistema, y el usuario, que lo utiliza para resolver un problema con la eficacia del especialista [SAM04].

Aunque muchas veces se usan los términos SBC y SE de manera indistinta, existe una distinción sutil entre ambos. Al hablar de SBC, en general, "se hace referencia a que lo más importante del sistema es el conocimiento que almacena y gestiona, es decir, se trata de una visión estructural del sistema" [UC311]. Sin embargo, "el término Sistema Experto se usa para referirse a un sistema que imita la actividad de un experto humano para resolver una determinada tarea en un dominio específico, es decir, se trata de una visión funcional del sistema" [UC311]. Por tanto, un SE puede catalogarse como un tipo particular de SBC, como se ilustra en la figura 1.

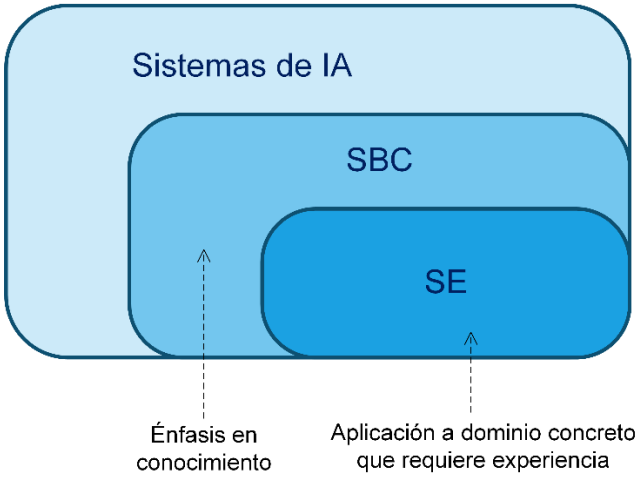

Figura 1. *Inteligencia Artificial, Sistemas Basados en Conocimiento y Sistemas Expertos*

En lo sucesivo, se usará el término "Sistema Experto".

Un SE, como vemos en la figura 2, está formado por dos elementos:

- Una **base de conocimiento**.
- Un **motor de inferencias**.

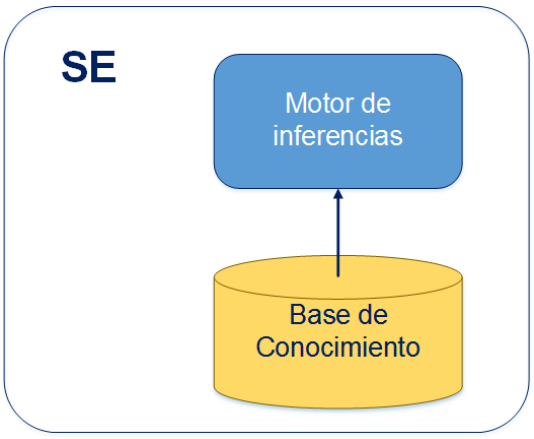

Figura 2. *Elementos de un Sistema Experto*

A continuación se explican dichos elementos en mayor detalle.

Base de conocimiento

La **Base de Conocimiento** (BC) debe representar de forma estructurada y entendible el conocimiento, con objeto de que éste pueda ser útil y relacionable para resolver los problemas que se plantean en el dominio de ejecución [MAR11].

El elemento responsable de representar la BC es la ontología. "Una **ontología** define los términos básicos y relaciones incluidos en el vocabulario de un dominio, así como las reglas para la combinación de los términos y relaciones para definir las extensiones del vocabulario" [NEC91].

Motor de inferencias

El otro elemento fundamental de un SE es el **motor de inferencias** o **razonador**, que está formado por algoritmos que gestionan la BC y facilitan la toma de decisiones del sistema. Esto es, permite obtener conclusiones a partir de la información recopilada en la BC (proceso de inferencia). Por tanto, podemos decir que el motor de inferencias es "el cerebro del SE" y razona sobre la BC.

Fases en el diseño de un SE

La construcción de un SE normalmente se desarrolla siguiendo las fases que están recogidas en la figura 3.

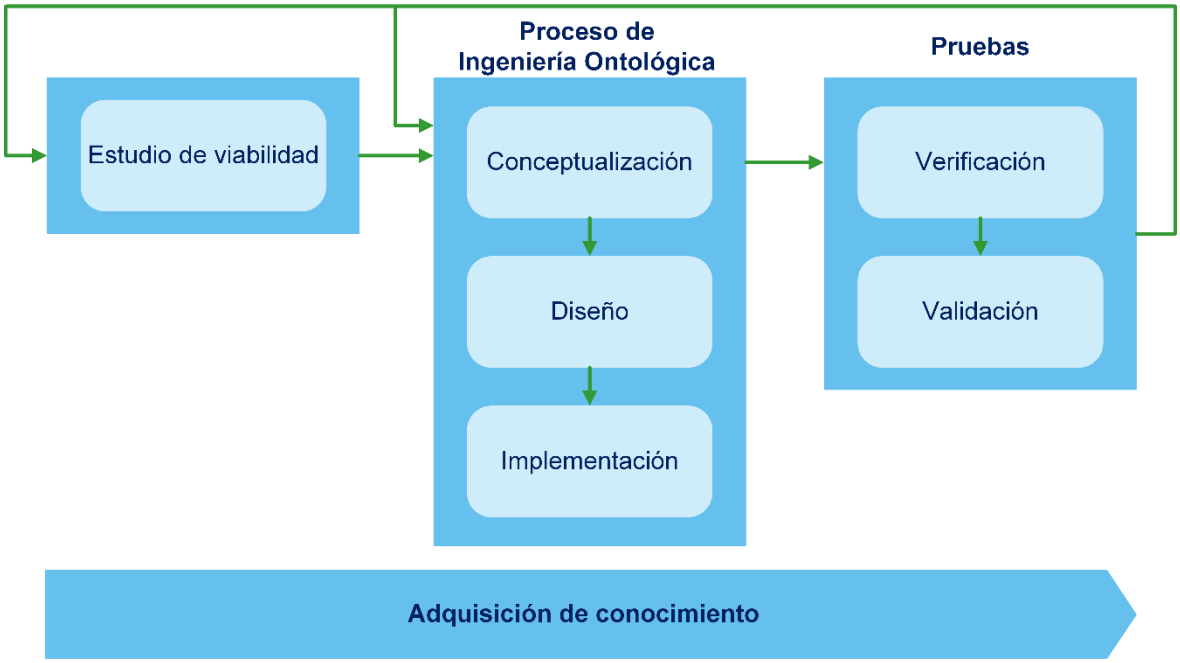

Figura 3. Proceso de construcción de un Sistema Experto

A continuación, se describen las tareas a realizar en cada una de las fases.

- **Estudio de viabilidad**. En esta fase se justifica que el problema planteado puede resolverse usando IC.

- **Adquisición de conocimiento**. Se extrae conocimiento de fuentes expertas en el dominio, tanto estáticas (revistas, libros, artículos, etc.) como dinámicas (entrevistas formales o informales, observación del campo real del experto, cuestionarios, etc.), con el objetivo de crear un modelo o diagrama conceptual del sistema [VAD07]. La adquisición del conocimiento se realiza durante todo el proceso de desarrollo del SE, pero de manera más exhaustiva al inicio del mismo.

- **Proceso de Ingeniería Ontológica**. En primer lugar se elabora un diagrama conceptual donde se muestran todos los conceptos del dominio y sus relaciones (**conceptualización**). Para ello se aplica alguna metodología, con el objetivo de que el diagrama resultante pueda ser legible y comprensible por cualquier persona. Dicho diagrama conceptual es lo que se conoce como ontología [VAD07].

  Una vez elaborada la ontología, se lleva a cabo una representación formal de la misma en un lenguaje que sea legible por un ordenador.

  Posteriormente, se inicia la fase de **diseño** del SE, donde se describe y se modela el sistema, antes de pasar a la fase de **implementación**, donde se eligen las plataformas y herramientas que se emplearan para construir el sistema diseñado.

- **Pruebas**. Una vez que ha sido formalizado el conocimiento en forma de ontología, tanto el ingeniero de conocimiento como los expertos en el dominio deben realizar pruebas. Esta fase incluye tanto la **verificación**, que es la comprobación de que el sistema realiza lo esperado en términos de funcionalidades, como la **validación**, en la que se comprueba la validez de los objetivos propuestos a la hora de diseñar el sistema.

Tal como muestra la figura 3, el proceso de construcción de un SE contempla la posibilidad de retroceder a una fase anterior, en caso de que se detecten inconsistencias o errores significativos que hagan necesaria la reestructuración del sistema.

2.1.2. Aplicaciones multicapa

La **arquitectura software** es el nivel más alto de abstracción de una aplicación o sistema informático y define: el conjunto de bloques o componentes software que forman el sistema, cómo estos componentes se conectan entre sí y sus interacciones.

Arquitectura de dos capas

Una de las posibles arquitecturas software es la arquitectura de dos capas, o **arquitectura cliente-servidor**, que es un modelo de aplicación en el que las tareas se reparten entre los proveedores de recursos o servicios, conocidos como servidores, y los demandantes, conocidos como clientes [WIK2]. Así, en esta arquitectura, uno o varios clientes realizan peticiones al servidor, que procesa las peticiones y envía una respuesta a cada petición, como vemos en la figura 4.

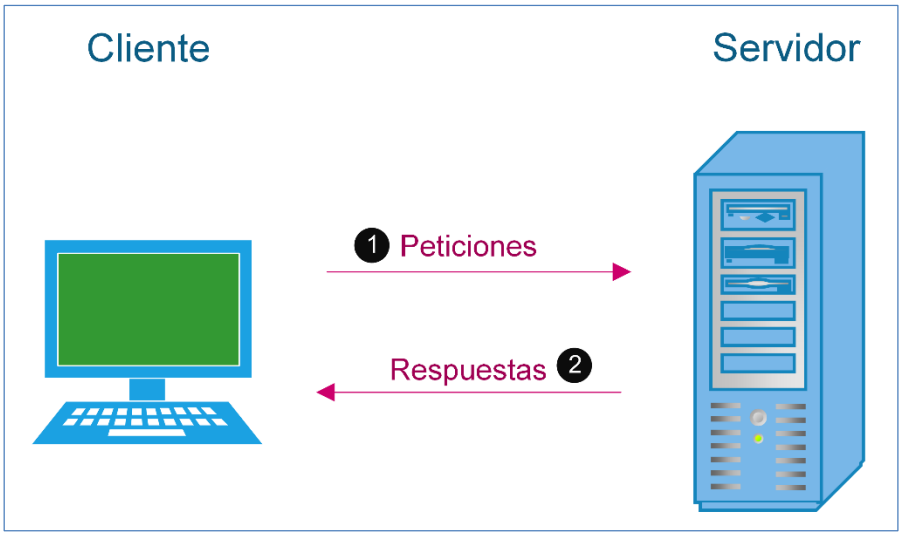

Figura 4. *Ejemplo de arquitectura cliente-servidor*

Aunque normalmente lo es, la separación entre cliente y servidor no ha de ser necesariamente física; puede existir una separación lógica, es decir, cliente y servidor pueden residir en una misma máquina. Si ambos residen en distintas máquinas, se comunicarán sobre una red de comunicaciones (por ejemplo, Internet).

La principal limitación de esta arquitectura es que, en una aplicación de cierta complejidad, o bien el cliente o bien el servidor, tendrán un tamaño considerable (*fat client* o *fat server*), dependiendo de si la lógica de la aplicación está implementada en el cliente o en el servidor, respectivamente.

Arquitectura de n-capas

La arquitectura de n-capas, concretamente la **arquitectura de tres capas**, supone una mejora de la arquitectura cliente-servidor, ya que permite superar varias de las limitaciones de ésta. Entre las mejoras que aporta la arquitectura de tres capas, destacan las siguientes:

- **Reducción de los cuellos de botella**. Los cuellos de botella consisten en una ralentización del servicio debido a la recepción de más peticiones de las que el

servidor puede gestionar en un cierto intervalo de tiempo. Esta anomalía puede llevar al cese del funcionamiento del servidor, lo que normalmente se conoce como "caída del servidor".

La reducción de este fenómeno redunda en un aumento del número de peticiones que el lado servidor de la aplicación puede gestionar.

- **Aumento de la escalabilidad**. La escalabilidad es la propiedad de una aplicación que indica su habilidad para adaptarse a cambios, que pueden implicar un crecimiento de la aplicación, sin perder calidad en los servicios ofrecidos.

  La división en tres capas hace que los roles en la aplicación se repartan más, de modo que la ésta pueda adaptarse con mayor facilidad a los cambios que se realicen sobre ella.

- **Aumento de la modularidad**. La modularidad "es la capacidad que tiene un sistema de ser estudiado, visto o entendido como la unión de varias partes que interactúan entre sí y que trabajan para alcanzar un objetivo común, realizando cada una de ellas una tarea necesaria para la consecución de dicho objetivo" [WIK3].

  La arquitectura de tres capas hace que la aplicación se puede subdividir más fácilmente en pequeñas partes (módulos) con unas funcionalidades más específicas. Cuanto más independientes sean los módulos entre sí, mayor será la facilidad para realizar cambios en la aplicación.

Vistas algunas de las bondades de la arquitectura en tres capas, veamos cuáles son estas tres capas y cuáles son sus funciones (ver figura 5):

1. **Capa de presentación**. Esta capa constituye la interfaz gráfica con el cliente, es decir, presenta la aplicación al usuario. También puede capturar información que el usuario introduzca y enviarla al servidor, pudiendo realizar ciertas comprobaciones acerca de la validez de los datos introducidos. Esta capa habitualmente sólo tiene comunicación directa con la capa de lógica.

2. **Capa de lógica** (o **de negocio** o **de lógica de negocio**). Tras recibir una petición del cliente, esta capa, normalmente ubicada en el servidor, realiza un procesamiento de la misma en base a unos algoritmos lógicos y envía una respuesta al cliente. Dicho procesamiento puede requerir comunicación con la capa de datos para recuperar o almacenar datos en la Base de Datos (BD).

3. **Capa de datos**: en esta capa se encuentra la BD, cuya función fundamental es almacenar persistentemente los datos manejados por la aplicación. Existirá un gestor de bases de datos que gestionará la BD.

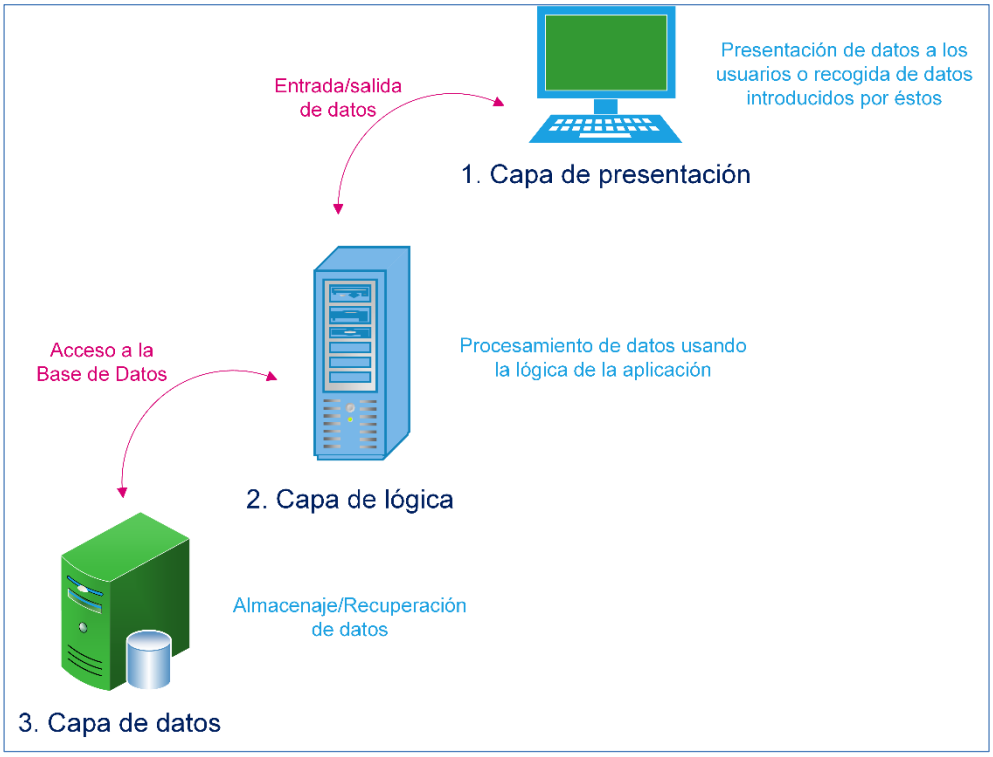

Figura 5. *Ejemplo de arquitectura de 3 capas*

Existen varias plataformas de software que permiten implementar aplicaciones siguiendo esta arquitectura. Una de ellas es la plataforma **Java EE** (*Java Platform, Enterprise Edition*; o Plataforma Java, Edición Empresarial).

Plataforma Java EE

Con el objetivo de simplificar el desarrollo de aplicaciones distribuidas, en 1999, Sun Microsystems lanzó la plataforma Java EE. La especificación Java EE tuvo un gran éxito y fue respaldada por importantes fabricantes de software [PUE09]. En la actualizad es propiedad de Oracle, y existen muchas herramientas disponibles, tanto gratuitas como de pago, para extender la plataforma o para simplificar el desarrollo de aplicaciones multicapa.

De modo general, Java EE constituye un **conjunto de especificaciones** que implementan una **arquitectura de n-capas sobre la web**. De modo más específico, Java EE provee al desarrollador de diferentes Interfaces de Programación de Aplicaciones (API, *Application Programming Interface*) y un entorno de ejecución para desarrollar y ejecutar aplicaciones en red a gran escala, que sean multicapa, escalables, fiables y más seguras [JEN10].

Las aplicaciones Java EE están construidas con componentes. Un **componente Java EE** es un una unidad de software independiente que se ensambla en una aplicación Java EE y que se comunica con otros componentes. Estos componentes están escritos en el lenguaje de programación Java y son compilados de la misma manera que cualquier programa en este lenguaje. De hecho, la plataforma Java EE está construida sobre la plataforma Java SE (*Java Platform, Standard Edition*; o Plataforma Java, Edición Estándar). Algunas de las diferencias entre los componentes Java EE y las clases Java SE son las siguientes [JEN10]:

- Los componentes Java EE están ensamblados en una aplicación Java EE, mientras que las clases Java SE no.
- Se comprueba que los componentes Java EE están bien formados y cumplen la especificación Java EE, mientras que los componentes Java SE no tienen que cumplir ninguna especificación más allá de utilizar correctamente la sintaxis del lenguaje de programación Java.

El enfoque de Java EE en componentes que cumplen una función específica **posibilita la reutilización** de los mismos. Es decir, si disponemos de un componente que resuelve un problema concreto de computación y aparece otro problema concreto cuya solución requiere del anterior componente, podemos utilizar el componente ya creado como parte de la solución del nuevo problema, sin necesidad de crear un nuevo componente.

Por otra parte, la existencia de la capa de lógica permite aumentar el aislamiento, o lo que es lo mismo, **disminuir el acoplamiento** entre la capa de presentación y la capa de datos. El acoplamiento se define como "el grado de interdependencia que hay entre los distintos módulos de un programa; lo deseable es que esta interdependencia sea lo menor posible, es decir, un bajo acoplamiento" [WIK4]. Un bajo acoplamiento aumenta la flexibilidad de la aplicación ante los cambios que se puedan producir en el mantenimiento o evolución de la misma [URI13].

En la figura 6 se pueden ver un ejemplo con dos aplicaciones Java EE. De esas dos aplicaciones, vamos a estudiar la segunda de ellas (*Java EE Application 2*, en la figura 6), por ser una aplicación web como Pegaso y Gades.

Dicha aplicación web está dividida en las siguientes capas:

- **Capa cliente** (*Client Tier*): está ubicada en la máquina del cliente (*Client Machine*).
- **Capa web** (*Web Tier*): se encuentra localizada en el servidor Java EE (*Java EE server)*.
- **Capa de negocio** (*Business Tier*): se haya en el servidor Java EE.

- **Capa del Sistema de Información Empresarial** (*EIS Tier*): se encuentra en el servidor de BD (*Database Server*).

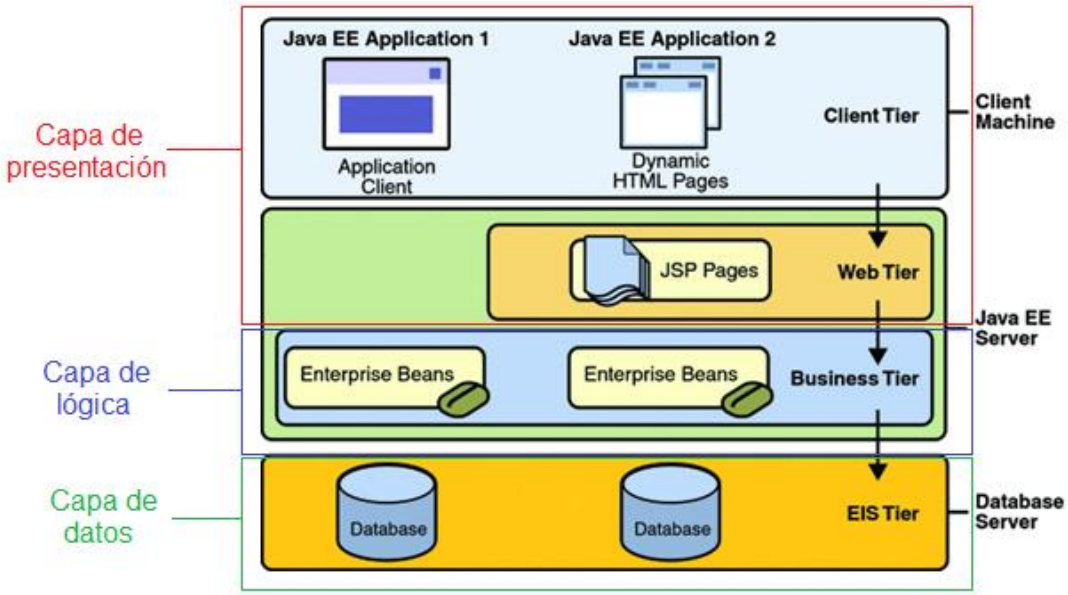

Figura 6. *Aplicaciones multicapa en Java EE (elaboración propia a partir de [JEN10])*

Aunque las aplicaciones Java EE pueden constar de cuatro capas, como es el caso expuesto en la figura 6, generalmente son catalogadas como "aplicaciones de tres capas" porque están distribuidas en tres localizaciones: máquinas de los clientes, máquina servidora Java EE y BD [JEN10].

A continuación, vamos a comparar las capas de la arquitectura en tres capas definidas anteriormente (capa de presentación, capa de lógica y capa de datos) con las capas definidas por Java EE. Describiremos, además, las operaciones que se llevan a cabo en dichas capas.

1. Capa de presentación

Está constituida por la "capa cliente" (situada en el cliente) y la "capa web" (residente en el servidor Java EE) definidas por Java EE. La "capa cliente" presenta al cliente la interfaz web de la aplicación a través del navegador web, permitiéndole acceder a la aplicación alojada en el servidor Java EE. La "capa web", por su parte, realiza el procesamiento necesario para generar el contenido que se muestra en el navegador del cliente.

La comunicación entre el cliente y el servidor se realizará mediante el protocolo HTTP (*Hypertext Transfer Protocol*), que es el protocolo empleado en la *World Wide Web* (WWW), comúnmente conocida como "la web". De ahí que hablemos de "aplicación web".

La secuencia de acciones que ocurren en esta capa es la siguiente: un cliente web enviará una petición HTTP al servidor web (por ejemplo, un usuario introducirá la URL del servidor en su navegador web) y el servidor, tras procesar la petición, generará una respuesta HTTP que enviará al cliente, y que contendrá una página web de uno de los dos tipos siguientes:

- **Página web estática**. Es un tipo de página destinada a mostrar una misma información permanentemente, sin permitir apenas interactividad del usuario con la misma, y generada con los lenguajes HTML (*HyperText Markup Language*) o XHTML (*eXtensible HTML*).

- **Página web dinámica**. Es un tipo de página generada en tiempo de ejecución en el servidor web (su contenido es generado en el momento en el que la solicitud es recibida en el servidor) por medio de la tecnología JSP (*JavaServer Pages*). Otorga mayores posibilidades de interactividad con el usuario (por ejemplo, listas desplegables cuyo contenido varíe en función del usuario que usa la aplicación).

En ambos casos se usan, además, las hojas de estilo CSS (*Cascading Style Sheets*) para dar formato a las páginas web por medio de unas reglas de estilo.

2. **Capa de lógica**

   Se corresponde con la "capa de negocio" definida por Java EE y se sitúa en el servidor Java EE, al que también se conoce como servidor de aplicaciones.

   Siguiendo con la aplicación de ejemplo, el servidor, a la hora de servir la respuesta HTTP al cliente, puede haber necesitado los servicios de los componentes o las clases incluidas en la capa de lógica para generar la respuesta. En ese caso, se habrán realizado una serie de operaciones siguiendo unos algoritmos lógicos para obtener la respuesta deseada.

3. **Capa de datos**

   Se corresponde con la "capa del Sistema de Información Empresarial" (ubicada en el servidor de BD) definida por Java EE. Esta capa se encarga de dar persistencia a los datos de la aplicación y provee de los mecanismos necesarios para acceder a estos datos.

Contenedores en Java EE

El servidor Java EE ofrece una serie de servicios en forma de contenedores para cada uno de los componentes de la aplicación. Los contenedores son interfaces entre los componentes y las funcionalidades específicas de bajo nivel que soporta el componente

[JEN10]. Los contenedores que nos interesan son los residentes en el lado del servidor, que se pueden ver recuadrados en color rojo en la figura 7.

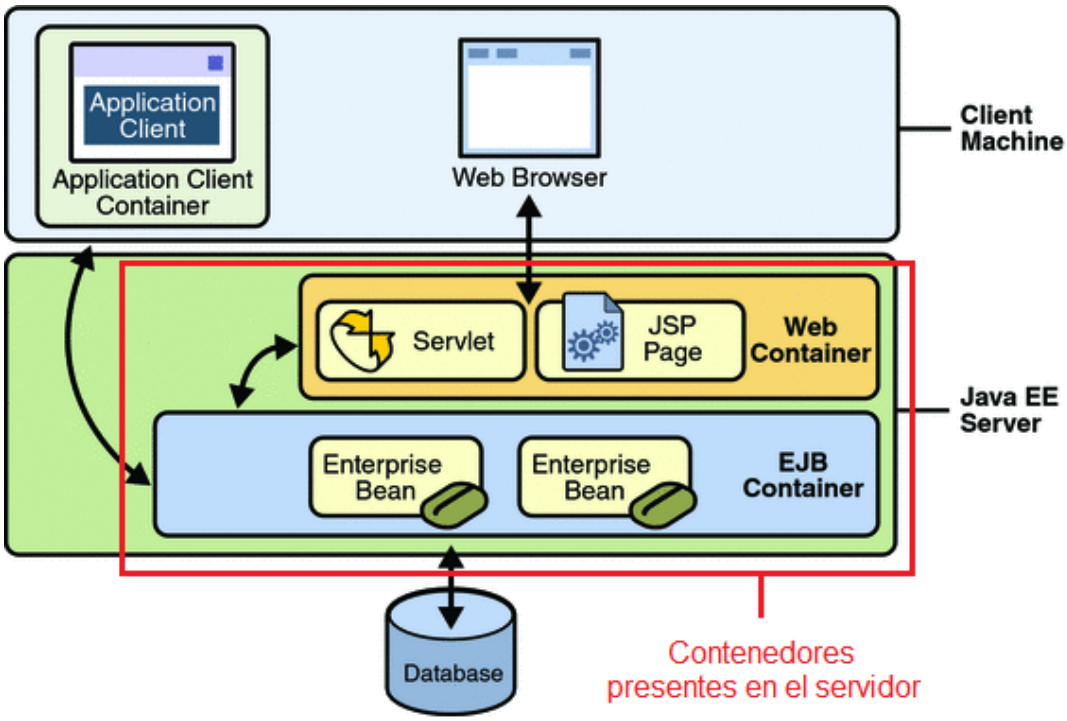

Figura 7. *Servidor Java EE y sus contenedores [JEN10]*

A continuación, exponemos dichos contenedores en mayor detalle y mencionamos algunas de las tecnologías que son propias de la especificación Java EE:

- El **servidor Java EE** (*Java EE Server*) constituye el entorno de ejecución (*runtime*) de las aplicaciones Java EE y provee el contenedor web y el contenedor de EJB.
- El **contenedor web** (*Web Container*) controla la ejecución de los siguientes componentes:
    - *Servlets*: clases Java que procesan peticiones y construyen respuestas dinámicamente.
    - *JavaServer Pages* (JSP): documentos basados en texto que se ejecutan como Servlets pero que permiten un enfoque más natural para la creación de contenido.
- El **contenedor de EJB** (*EJB Container*) controla la ejecución de:
    - *Enterprise JavaBeans* (EJB): son componentes de lógica ubicados en el servidor.

Tanto el contenedor web como el contenedor de EJB, y los componentes de ambos, se ejecutan sobre el servidor Java EE. Los componentes no se comunican directamente, sino que lo hacen a través de protocolos ofrecidos por los contenedores.

2.2. Sistemas Pegaso y Gades

Pegaso y Gades son dos aplicaciones web multicapa que constituyen dos SE. Por ello, vamos a abordar el estudio de ambos sistemas desde dos puntos de vista complementarios:

- Pegaso y Gades como Sistemas Expertos (apartado 2.2.1).
- Pegaso y Gades como aplicaciones multicapa (apartado 2.2.2).

Además, explicaremos las funcionalidades de ambos sistemas (apartado 2.2.3), el funcionamiento de Pegaso (apartado 2.2.4), y por último, el funcionamiento de Gades (apartado 2.2.5).

2.2.1. Pegaso y Gades como Sistemas Expertos

Desde el punto de vista de la IC, Pegaso y Gades son dos SE en el ámbito de la e-Salud que apoyan a distintos profesionales en la detección precoz de trastornos del lenguaje en niños. Como todo SE, Pegaso y Gades constan de una BC y un motor de inferencias:

- **Base de Conocimiento**. Una vez definida la ontología, ésta se formalizó mediante Lógica Descriptiva, DL (*Description Logic*), haciendo uso del **lenguaje de ontologías web OWL DL** (*Web Ontology Language Description Logic*). Este lenguaje establece un conjunto de reglas que permiten describir y representar conocimiento del dominio de la neuropediatría, en este caso [MAR11].

  Como apoyo a la implementación de la ontología se utilizó el editor de ontologías Protégé, software libre[3] y de código abierto[4], que permite obtener un archivo en el cual se almacena la ontología.

[3] **Software libre** (*free software*) es aquel que "respeta las libertades esenciales del usuario: la libertad de utilizarlo, ejecutarlo, estudiarlo y modificarlo, y de distribuir copias con o sin modificaciones. Es una cuestión de libertad y no de precio, por lo tanto piense en «libertad de expresión» y no en «barra libre»" [STA14]. Esencialmente, es una visión del software desde el punto de vista de la ética.

[4] **Software de código abierto** (*open-source software*) "es el software publicado bajo una licencia de software compatible con la *Open Source Definition*. Esto permite a los usuarios utilizar, cambiar, mejorar el software y redistribuirlo, con o sin modificaciones" [WIK5]. Es una visión del software desde el punto de vista técnico esencialmente.

- **Motor de inferencias** o **razonador**. Pegaso y Gades emplean el razonador Pellet, que se basa en Java y es de código abierto.

  En cada evaluación del nivel de adquisición del lenguaje realizada, el motor de inferencias analiza la información introducida por el usuario, contrastándola con el conocimiento existente en la BC, y si el nivel de adquisición no es el normal (el correspondiente a sus meses de edad), tomará una decisión (derivar al especialista correspondiente, adelantar una visita, etc.) [MAR11].

2.2.2. Pegaso y Gades como aplicaciones multicapa

Pegaso y Gades son dos **aplicaciones web** que siguen la **arquitectura de tres capas** y emplean varias de las tecnologías usadas en Java EE, como se decidió y justificó en [URI13]. La arquitectura de Pegaso y Gades (figura 8) es una simplificación de la explicada en el apartado 2.1.2 para la plataforma Java EE (ver figura 6).

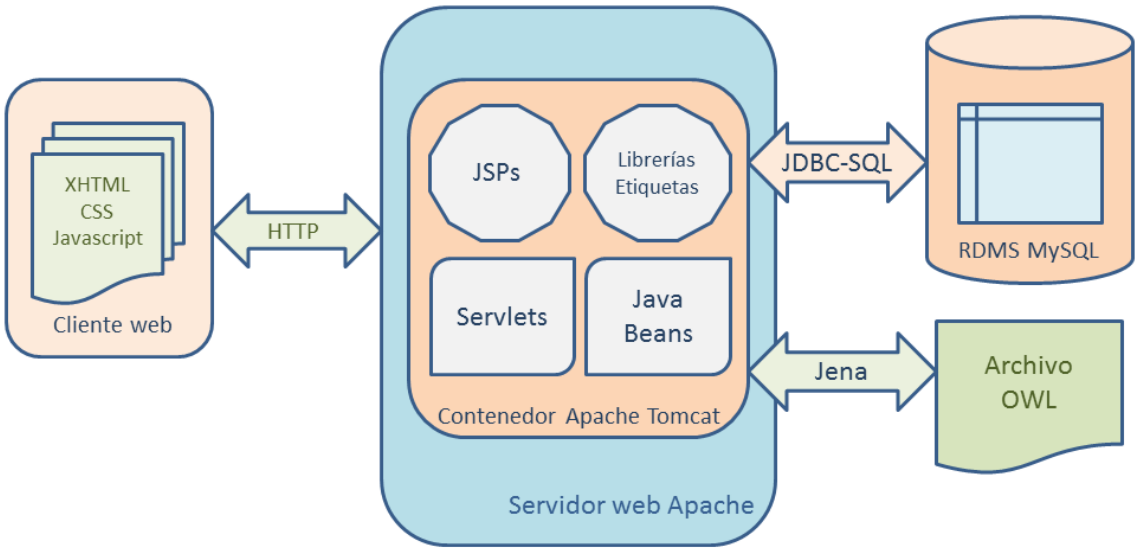

Figura 8. *Arquitectura de las aplicaciones Pegaso y Gades [URI13]*

El **servidor** elegido es **Apache Tomcat**, software libre y de código abierto. Apache Tomcat, en las versiones utilizadas hasta ahora (hasta la 7.0.50) no es exactamente compatible con la especificación Java EE. Sin embargo, la versión 8.0.9 (publicada en mayo de 2014) sí cumple la especificación Java EE. En cualquier caso, las versiones anteriores de Tomcat proveen el soporte necesario a las tecnologías que se emplean en los sistemas Pegaso y Gades.

Apache Tomcat es un servidor web con soporte para Servlets y JSP. Como el único contenedor necesario es el contenedor web y Apache tiene su propio contenedor web Java [URI13], no existe la necesidad de tener un servidor de aplicaciones.

Por otra parte, en la capa de datos se usa el **sistema gestor de base de datos relacional MySQL**, software libre y de código abierto. Este gestor, cuyo uso es muy extendido, proporciona las herramientas y mecanismos necesarios para añadir, borrar, consultar y modificar los datos de la aplicación.

El **cliente web**, por su parte, recibe páginas XHTML estáticas y páginas JSP, pero también ejecuta código *JavaScript* y AJAX (*Asynchronous JavaScript And XML*) para llevar a cabo ciertas tareas en el lado del cliente que no pueden ser llevadas a cabo por la tecnología JSP.

Arquitectura de componentes

Como explicamos en el apartado 2.1.2, las aplicaciones Java EE se dividen en componentes. Así pues, a continuación describimos la arquitectura de componentes de Pegaso y Gades.

1. Capa de presentación

En el lado del **servidor**, hay componentes web de dos clases diferentes [URI13]:

- **JSP** (*JavaServer Pages*): construyen las páginas que visualiza el usuario (vistas de usuario). Son páginas dinámicas que se ejecutan en el servidor y generan una respuesta en forma de página web, similar a una página XHTML.
- **Servlets**: son clases Java especializadas en el tratamiento de peticiones y respuestas HTTP. Poseen funciones que permiten implementar respuestas para los diferentes métodos de dicho protocolo. Implementan el tratamiento de peticiones y de respuestas que requieren la ejecución de cierta lógica.

El **tratamiento de peticiones y respuestas** en el servidor web de las aplicaciones Pegaso y Gades se muestra en la figura 9 y se explica en mayor detalle a continuación [URI13]:

1. El cliente web (navegador) origina una petición destinada al servidor web donde reside la aplicación.
2. La petición, al llegar al servidor, se encapsula dentro de un objeto Java de clase *HttpServletRequest*. Este objeto, que contiene toda la información que viaja en la petición, se pasa a un componente web que procesa la petición.
3. El componente web hace llamadas a métodos de componentes JavaBean que implementan la parte del procesado correspondiente a otras capas (lógica y acceso a datos).

4. Los componentes JavaBean realizan el procesado correspondiente a la capa a la que pertenecen (por ejemplo, los de acceso a datos accederán a la base de datos). Al finalizar el procesado devuelven información a los componentes web de la capa de presentación.
5. El componente web correspondiente hace el tratamiento de los datos y genera una respuesta para el cliente web. Esta respuesta se encapsula en un objeto Java de la clase *HttpServletResponse*.
6. El servidor web envía la respuesta HTTP con la información que encapsulaba el objeto *HttpServletResponse*.

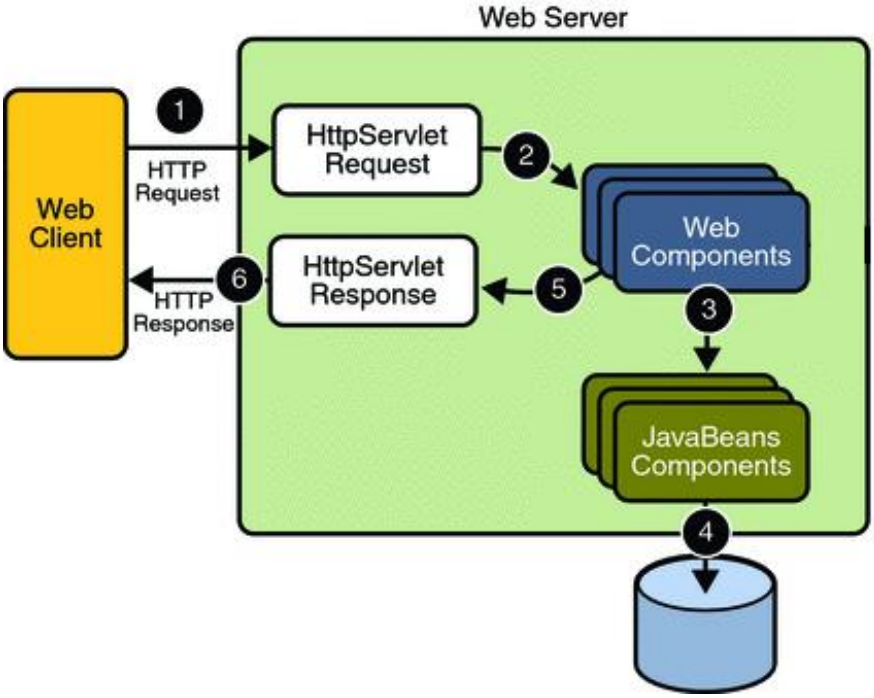

Figura 9. *Tratamiento de peticiones y respuestas HTTP en una aplicación web Java [JEN10]*

En el tratamiento de una petición pueden intervenir varios componentes web que repartan sus responsabilidades en el tratamiento de la petición. De hecho, existen una serie de **patrones de diseño** que constituyen soluciones típicas a problemas encontrados habitualmente en las diferentes capas de procesamiento de una aplicación web Java EE [URI13].

En la capa de presentación se han usado dos patrones de diseño diferentes [URI13]:

- **Patrón de ayudante de vista**: su función es simplificar el acceso a la lógica de acceso a datos para las vistas de usuario (JSP). Los componentes que implementan este patrón son Servlets.

- **Patrón de controlador frontal**: su función es centralizar el procesado de peticiones originadas en las vistas de usuario (JSP). Los componentes que implementan este patrón son Servlets también.

En definitiva, existen tres roles para los dos componentes presentes en la capa de presentación (JSP y Servlets):

- Vistas para los usuarios (JSP).
- Ayudantes de vista (Servlets).
- Controlador frontal (Servlets).

Las vistas y los ayudantes de vista comparten información mediante un **contexto de colaboración** constituido por un objeto denominado **sesión**. Este objeto sesión es gestionado por el contenedor web y está asociado a cada una de las sesiones abiertas en el sistema por los usuarios del mismo. Es decir, cada usuario que interactúa con Pegaso o Gades tiene asociada en el lado del servidor una información relacionada con su actividad. Esta información se almacena y se recupera del objeto sesión por parte de los componentes web de la capa de presentación [URI13].

El objeto sesión almacena la información en forma de múltiples objetos JavaBean o de colecciones de objetos JavaBean. Cada uno de estos objetos, almacenados en la sesión, tiene asociado un nombre único que permitirá el acceso al mismo. La pareja "nombre de objeto-objeto" que se almacena en la sesión se denomina **atributo de sesión**. En [URI13] se definen los atributos de sesión usados en Pegaso y Gades. En este PFG también se definirán algunos nuevos atributos de sesión.

2. Capa de lógica

Esta capa se encarga de realizar el procesado asociado a las funcionalidades de las aplicaciones Pegaso y Gades. Se sitúa entre la capa de presentación y la de acceso a datos. Luego, esta capa genera información para ser presentada al usuario o trata la información que éste ha introducido (interacción con la capa de presentación), y se comunica con la capa de datos para almacenar datos en la BD o recuperarlos de ésta (interacción con la capa de datos) [URI13].

En la capa de lógica se usan los siguientes patrones de diseño [URI13]:

- **Patrón de delegado de negocio** (*business delegate*): reduce el acoplamiento entre capas (ver apartado 2.1.2). Los componentes que implementan este patrón son los

objetos **JavaBean de lógica de negocio** (no confundir con EJB), que son también denominados controladores de lógica.

- **Patrón de objeto de transferencia**: son objetos JavaBean cuya utilidad es encapsular datos a transferir entre capas.
- **Patrón de manejador de lista de valores**: su utilidad es iterar eficientemente en una lista amplia de datos.

Además, en esta capa se utilizan los siguientes API:

- **JFreeChart**: este API, software libre y de código abierto, permite dibujar una amplia gama de gráficas (de barras, de líneas, de barras, circulares, etc.). JFreeChart soporta diferentes formatos de salida para las gráficas (PNG, JPEG, PDF, etc.), además de componentes Swing (librería gráfica para Java).
- **Apache Jena**: está compuesto por varios API que permiten el acceso y tratamiento de ontologías desde las aplicaciones. Es una herramienta de código abierto que puede trabajar con el lenguaje OWL, con lo cual permite que Gades y Pegaso puedan usar el archivo de extensión .owl que contiene la ontología.

3. **Capa de datos**

Esta capa procesa la información que se desea almacenar u obtener de la BD y en ella se usa el siguiente patrón de diseño [URI13]:

- **Patrón de objeto de acceso a datos**: su función es abstraer y encapsular los mecanismos de acceso a datos. Nos referimos a los objetos que implementan este patrón como DAO (*Data Access Object*).

Por tanto, los métodos que realizan los accesos a la BD se encapsulan en componentes DAO, que son clasificados como objetos JavaBean. Los DAO hacen uso del API **Java DataBase Connectivity** (JDBC) para acceder o almacenar los datos en la BD. Así, este API permite ejecutar y enviar a la BD sentencias SQL (*Structured Query Language*) para consultar, insertar, actualizar o borrar datos.

La capa de datos se comunica con la capa de lógica intercambiando objetos JavaBean de transferencia de datos o colecciones de los mismos [URI13].

Como puede deducirse de lo descrito hasta el momento, existen tres tipos de objetos JavaBean:

- Los controladores de lógica.

- Los objetos de transferencia de datos.
- Los objetos de acceso a datos (DAO).

Por otra parte, en la capa de datos también se encuentra el **archivo** de extensión **.owl** que contiene la ontología usada en el proceso de inferencia del sistema. Este archivo, como hemos explicado anteriormente, es accedido desde la capa de lógica por los API de Apache Jena.

Resumen de la arquitectura de componentes

La arquitectura basada en componentes de las aplicaciones multicapa Pegaso y Gades se resume en la figura 10.

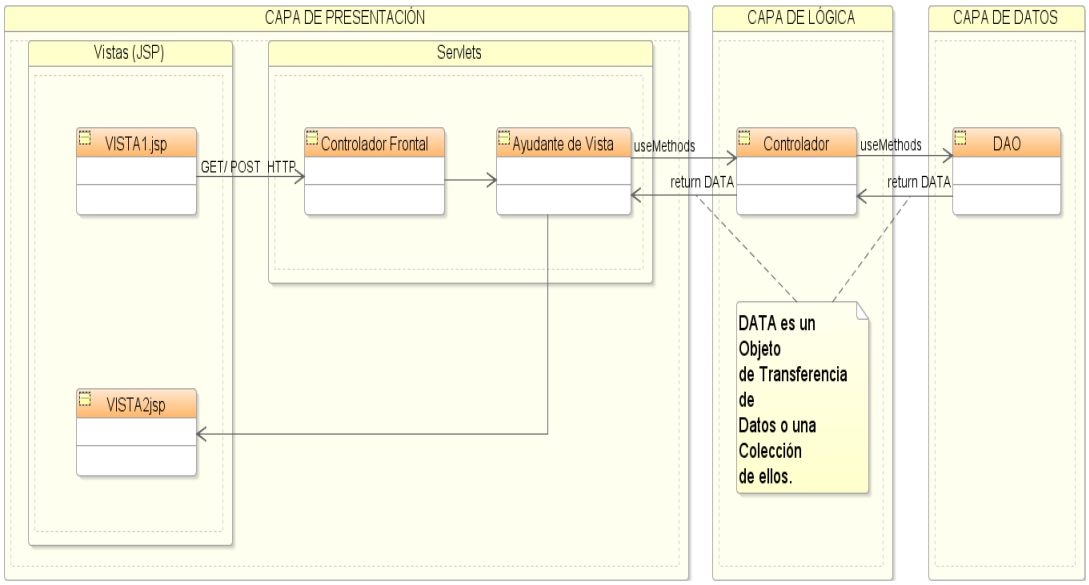

Figura 10. *Arquitectura de las aplicaciones multicapa Pegaso y Gades [URI13]*

La interacción mostrada en la figura 10 se explica resumidamente en los párrafos siguientes.

En la capa web se distinguen las vistas (páginas JSP) del controlador frontal (clase *MedicosFrontController*) y de los ayudantes de vista (Servlets). El controlador frontal recibe la petición y, tras evaluarla, la dirigirá al ayudante de vista correspondiente, además de encargarse de labores de autenticación, autorización y gestión de usuarios médicos. Será el ayudante de vista quien realizará el procesado adicional e invocará a los métodos de los controladores de lógica (clases *EvaluacionLenguajeController* y *MedicosController*).

Cada controlador de lógica (o delegado de negocio) realizará su procesado y usará los métodos del objeto de acceso a datos (DAO) para obtener o enviar datos a la BD.

El DAO, tras acceder a la BD, devolverá los datos al controlador de lógica en forma de JavaBean (objeto de transferencia) o de una colección de los mismos. Estos datos se podrán devolver a un ayudante de vista de la capa de presentación que los redirigirá para su presentación a la vista correspondiente.

2.2.3. Funcionalidades de Pegaso y Gades

Las aplicaciones Pegaso y Gades difieren en los usuarios a los que están dirigidas, pero sus funcionalidades son las mismas, como podemos ver en la figuras 11 y 12, que corresponden a las interfaces web de Pegaso y Gades, respectivamente. A través de estas interfaces se puede acceder a ambos sistemas.

Figura 11. Interfaz web de Pegaso (página de acceso)

Figura 12. Interfaz web de Gades (página de acceso)

Concretamente, las funcionalidades de ambos sistemas son las siguientes:

- **Evaluación del lenguaje**. El usuario responde a una serie de preguntas relacionadas con el nivel de adquisición del lenguaje del niño y, a partir de las respuestas introducidas y accediendo a la BC, el motor de inferencias infiere, es decir, toma una decisión, que puede ser:

- Un resultado normal.
- Una modificación del calendario de visitas del niño al pediatra (aviso).
- Una derivación a un cierto especialista (alarma).

Tanto si la decisión es un aviso como si es una alarma, se da al usuario la posibilidad de imprimir un documento con la decisión del sistema para entregárselo al familiar del niño que corresponda.

- **Consulta de resultados**. El usuario puede consultar las respuestas y los resultados de las evaluaciones del lenguaje realizadas con anterioridad a la fecha de esta consulta.
- **Consulta de estadísticas**. El usuario puede consultar diferentes estadísticas referentes a las evaluaciones del lenguaje realizadas con anterioridad a la fecha de esta consulta. Esta funcionalidad ha sido implementada en el transcurso del presente proyecto y se explica en mayor detalle en el capítulo 3.

Ambos sistemas poseen, además, una serie de **funcionalidades administrativas**, a las cuales se accede mediante la página presentada en la figuras 13 (la de Gades es prácticamente igual, por lo que no se incluye).

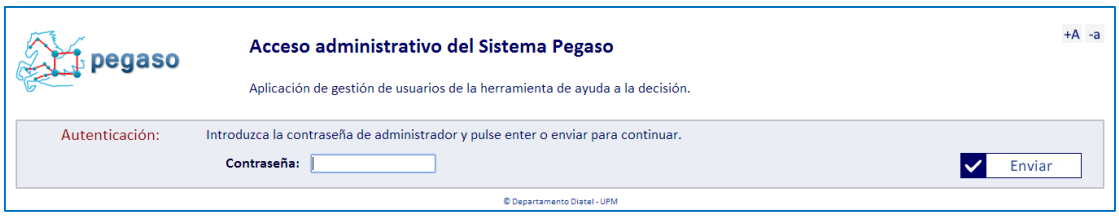

Figura 13. *Interfaz web de Pegaso (acceso administrativo)*

Estas funcionalidades administrativas permiten realizar las siguientes tareas:

- **Consultar** todos los **médicos** o consultar médicos siguiendo un criterio: por centro médico, por especialidad, por apellidos o por número de colegiado.
- Añadir o **dar de alta** nuevos **médicos**.
- Eliminar o **dar de baja médicos**.

2.2.4. Descripción de las funcionalidades de Pegaso

En este apartado explicaremos las **funcionalidades de partida** del sistema Pegaso. Se excluye, por tanto, la funcionalidad de consulta de estadísticas.

La figura 14 resume la interacción entre los actores implicados en Pegaso.

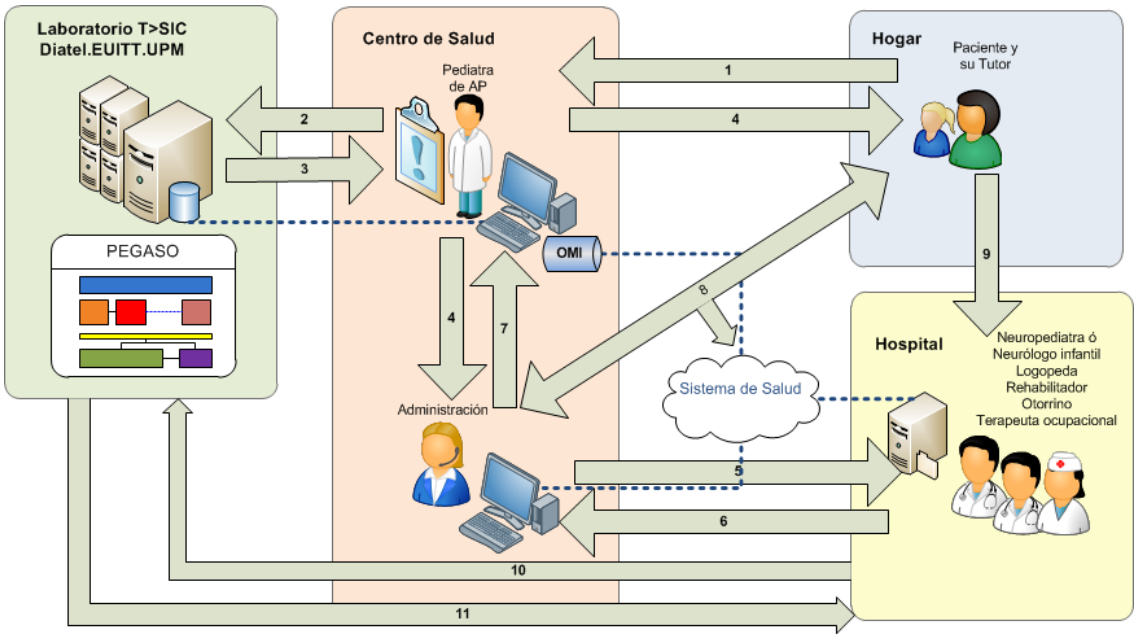

Figura 14. Arquitectura funcional del sistema Pegaso [MAR13]

Esta interacción se explica en detalle a continuación [MAR13]:

1. El niño acude al pediatra de familia acompañado de un miembro de su familia.

2. El pediatra de AP decide utilizar el sistema desarrollado para evaluar si existe algún trastorno del lenguaje en el niño, en cuyo caso se realizará la derivación precoz al especialista correspondiente o bien se adelantará la próxima visita del niño con objeto de realizar una nueva evaluación. De este modo, el pediatra inicia su interacción con el sistema usando la funcionalidad de **evaluación del lenguaje**, la cual se describe en detalle a continuación.

   El primer paso que da el pediatra es introducir su nombre de usuario y su contraseña y, una vez que el servidor comprueba que el usuario existe y la contraseña es correcta (proceso de autenticación), el sistema solicita al pediatra los datos generales del niño. Podemos ver un ejemplo en la figura 15.

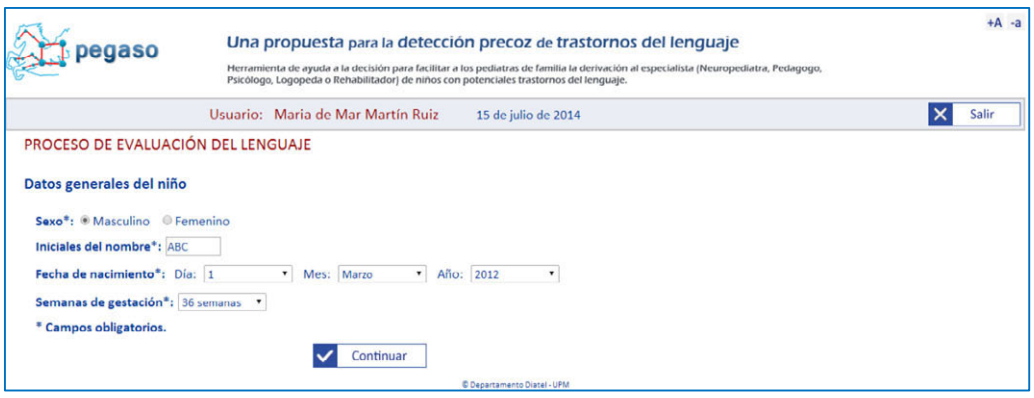

Figura 15. Pantalla inicial de la funcionalidad de evaluación del lenguaje en Pegaso

A continuación, tras pulsar en "Continuar", el sistema presenta al pediatra las preguntas relativas al grado de adquisición del lenguaje del niño. En la figura 16 podemos ver las preguntas correspondientes al niño de ejemplo (ver figura 15).

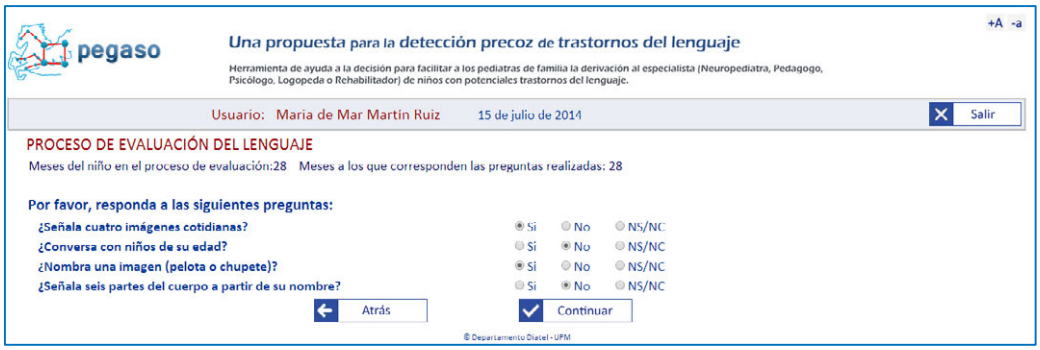

Figura 16. *Pantalla con preguntas de la funcionalidad de evaluación del lenguaje en Pegaso*

3. Una vez que las preguntas han sido respondidas, y tras pulsar en "Continuar", el sistema devuelve el resultado al pediatra. Existen dos posibilidades:

   - El resultado es normal, en cuyo caso el niño vuelve a su casa sin modificar el curso normal de visitas al pediatra.
   - El resultado modifica el calendario de visitas del niño al pediatra, o bien hay que derivar al especialista pertinente del hospital correspondiente.

   En el caso del niño de ejemplo, la decisión del sistema, a partir de las respuestas introducidas (ver figura 16), es la que aparece en la figura 17: "Derivar al neuropediatra" y "Derivar al equipo de Atención Temprana".

   Además, como también se puede ver en la figura 17, se deben introducir una serie de datos adicionales acerca del niño, como el peso del mismo al nacer. También es obligatorio que el pediatra indique si realizará la acción propuesta en el sistema o no. Tras introducir estos datos, se puede guardar la evaluación realizada pulsando en "Guardar".

Figura 17. Pantalla de resultados de la funcionalidad de evaluación del lenguaje en Pegaso

Por último, el sistema confirma al pediatra si la operación se ha realizado con éxito o no. En el ejemplo propuesto, como podemos ver en la figura 18, el proceso ha finalizado correctamente. Tras pulsar en "Continuar", el sistema vuelve a la pantalla inicial.

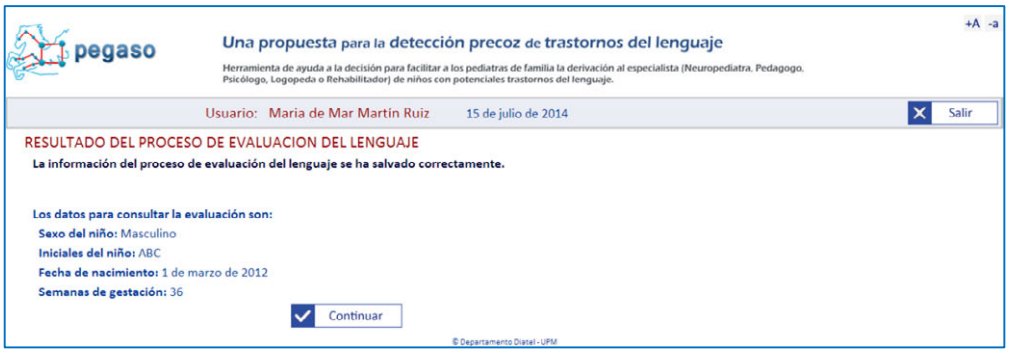

Figura 18. Pantalla final de la funcionalidad de evaluación del lenguaje en Pegaso

4. y **5.** Se realiza la petición de cita con el especialista correspondiente del hospital.

6. Se recibe respuesta a la petición de cita con el hospital.

7. Los datos de la cita con el especialista los recibe el pediatra.

8. Los datos de la cita llegan al niño y a su familia.

9. El niño acude al especialista correspondiente.

10. y **11.** El especialista quiere consultar la respuesta que el sistema produjo para el caso de estudio correspondiente. Esto se corresponde con funcionalidad de **consulta de resultados**, que se describen en detalle a continuación.

Una vez que el usuario se ha autenticado, se le solicitan los siguientes datos: primero, el centro al que pertenece el niño y el pediatra que realizó la evaluación del lenguaje; y, después, los datos generales del niño. Podemos ver un ejemplo en la figura 19.

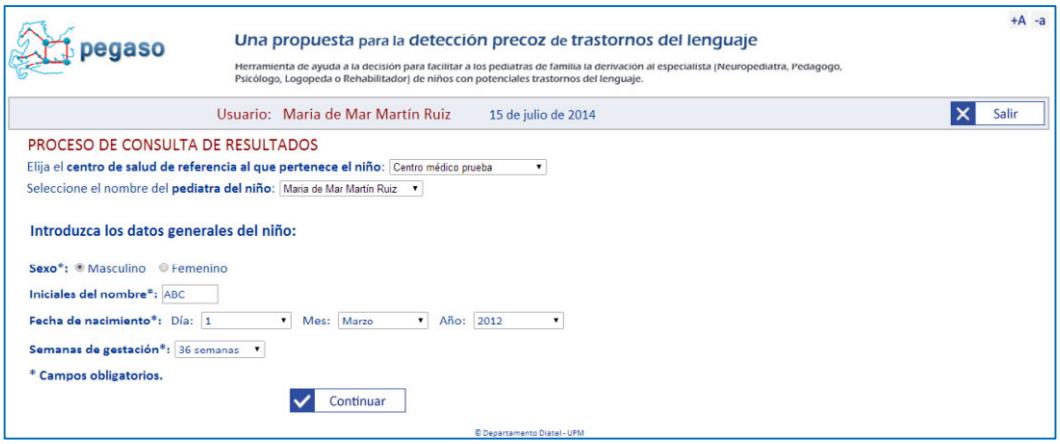

Figura 19. Pantalla inicial de la funcionalidad de consulta de resultados en Pegaso

Tras pulsar en "Continuar", el sistema comprueba si existen evaluaciones correspondientes al paciente introducido. En caso afirmativo, muestra los resultados del proceso de evaluación del lenguaje, si solo hay una evaluación; o, si hay varias evaluaciones para el mismo paciente, solicita al usuario que escoja una de las evaluaciones. Podemos ver el primer caso (sólo hay una evaluación) en la figura 20.

Además, existe la posibilidad de que los usuarios puedan valorar la decisión propuesta por el sistema un máximo de una vez por evaluación.

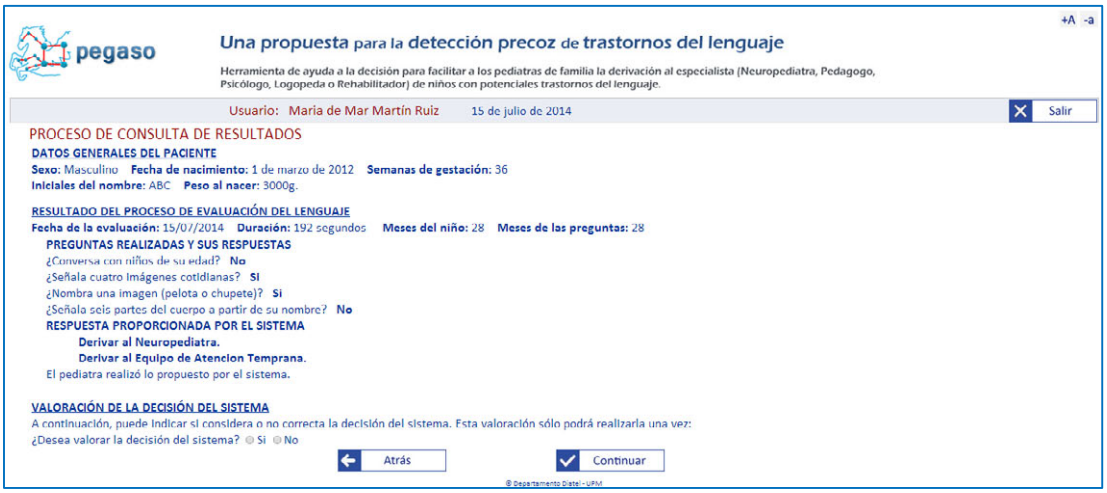

Figura 20. Pantalla final de la funcionalidad de consulta de resultados en Pegaso

Finalmente, pulsando en "Continuar" podemos volver a la pantalla inicial de Pegaso.

2.2.5. Descripción de las funcionalidades de Gades

Este apartado presenta la funcionalidad de evaluación del lenguaje del sistema Gades. El proceso, desde una perspectiva global, es ligeramente diferente al de Pegaso, debido a que ambos sistemas se aplican en un ámbito diferente. No obstante, la funcionalidad de consulta de resultados es igual en uno y otro sistema. Recuerde que la funcionalidad de consulta de estadísticas no se explica porque no forma parte del sistema de partida, sino que se desarrolla como parte de este PFG.

La figura 21 muestra la interacción entre los actores implicados en Gades [MAR14]. Esta interacción se explica en detalle a continuación:

1. El profesional educativo que corresponda (educador infantil o logopeda del colegio) decide comenzar el proceso de evaluación del lenguaje de sus alumnos utilizando el sistema Gades. Para esto debe obtener un cuestionario, en papel, con las preguntas correspondientes a los hitos de desarrollo que están almacenados en la BC y que debe observar según la edad del niño en meses.

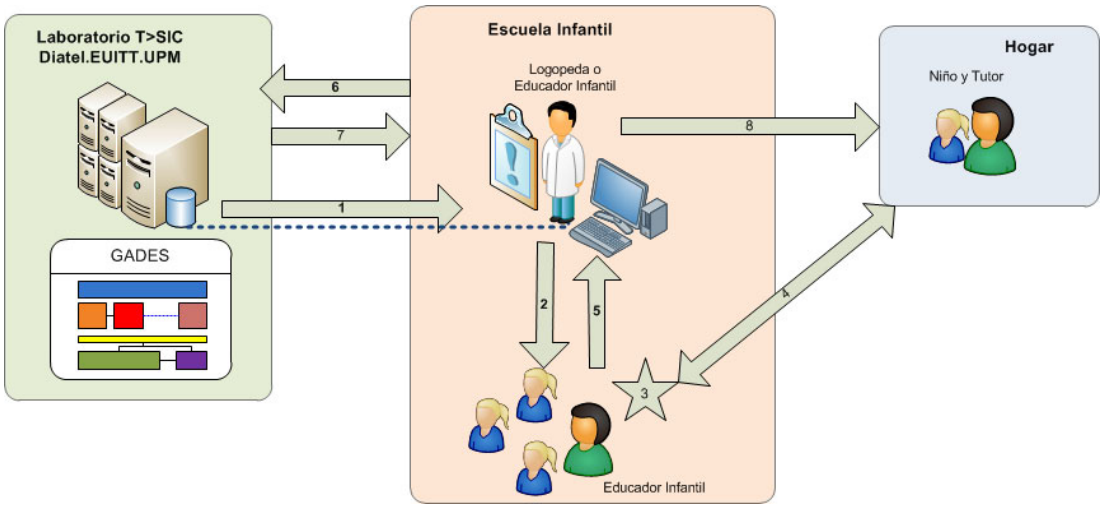

Figura 21. *Arquitectura funcional del sistema Gades*

2. El usuario le proporciona al educador infantil el cuestionario con la observación que debe realizar para cada niño.

3. El educador infantil realizará la observación del nivel de adquisición del lenguaje de sus alumnos. Esta observación durará una o dos semanas dependiendo del tiempo que el educador necesite para responder a todas las preguntas.

4. Es posible que para responder a alguna de las preguntas el educador tenga que recurrir a los padres, puesto que en algunas ocasiones el niño se comporta de forma distinta en el colegio y en casa.

5. Cuando el educador finaliza todos los cuestionarios de sus alumnos, se los devuelve al usuario encargado de realizar la evaluación del lenguaje utilizando Gades.

6. **y 7.** El usuario interactúa con el sistema Gades realizando el proceso de **evaluación del lenguaje** de todos los niños que participan en el estudio.

8. Cuando el resultado obtenido para algún alumno sea alarmante, el centro educativo valorará la idoneidad de informar a los padres con objeto de realizar un diagnóstico completo del posible trastorno detectado por Gades.

3. Descripción de las soluciones propuestas

En este capítulo se describen y justifican los problemas propuestos en este PFG, las diferentes alternativas que pueden resolverlos y las soluciones escogidas y puestas en marcha.

3.1. Incompatibilidad de Pegaso y Gades con Java SE 7

Comenzaremos nuestro estudio con una introducción a la tecnología Java SE (apartado 3.1.1), antes de dar paso a una descripción de las versiones de Java SE empleadas hasta la fecha en Pegaso y Gades (apartado 3.1.2). Tras ello, se describirá el error que se produce al inferir con Java SE 7 (apartado 3.1.3) y, por último, se explicará la solución escogida para solucionarlo (apartado 3.1.4).

3.1.1. Introducción a Java SE

Las aplicaciones Pegaso y Gades, como se explica en el apartado 2.2.2, usan el lenguaje de programación Java en la capa de lógica, ubicada en el servidor web. El lenguaje Java está descrito en la Especificación del Lenguaje Java, *Java Language Specification* (JLS), publicada por Oracle. La última versión del lenguaje Java publicada, en marzo de 2014, es Java SE 8. Cada cierto tiempo se publican actualizaciones (*updates*) de las versiones que aún son mantenidas (las versiones más antiguas van cayendo en desuso y dejan de publicarse actualizaciones). Por ejemplo, la última actualización para Java SE 7 es, hasta la fecha, "Java SE 7 Update 65".

Las nuevas versiones de Java SE incluyen nuevas y mejoradas funcionalidades, especialmente en el lenguaje Java en sí, mientras que las sucesivas actualizaciones de cada versión arreglan *bugs* existentes en la versión a la que correspondan. El uso de las últimas versiones de Java SE es recomendable debido a que:

- Las mejoras de cada nueva versión habitualmente mejoran el rendimiento de las aplicaciones, aunque desde el punto de vista del usuario final pueda no ser percibido.
- Se brinda al desarrollador la posibilidad de usar cualquiera de las nuevas funcionalidades introducidas en las nuevas versiones.

Por tanto, es deseable que la versión de Java SE usada en el servidor sea actualizada cada cierto tiempo.

3.1.2. Versiones de Java SE usadas

La versión de Java SE usada por las aplicaciones Gades y Pegaso hasta el inicio de este PFG ha sido **Java SE 6**. Esto quiere decir que en la máquina que aloja el servidor web Apache Tomcat están instaladas las siguientes herramientas:

- **JDK 6** (*Java SE Development Kit*). El JDK es el conjunto de herramientas para desarrollar, depurar[5] y monitorizar aplicaciones Java. No es imprescindible tenerlo instalado para ejecutar aplicaciones Java, sí para desarrollarlas. Cuando nos referimos a actualizar Java SE 6, lo que realmente se actualiza es el JDK.

- **JRE 6** (*Java SE Runtime Environment*). El JRE es el conjunto de utilidades necesarias que permiten la ejecución de programas Java. Está formado por la Máquina Virtual de Java o JVM (*Java Virtual Machine*), un conjunto de librerias Java y otros componentes necesarios para que una aplicación escrita en lenguaje Java pueda ser ejecutada [WIK6]. El JRE actúa como un "intermediario" entre el sistema operativo y Java. Por tanto, es imprescindible tenerlo instalado para ejecutar aplicaciones Java. Dado que está incluido en el JDK, no hace falta descargarlo separadamente.

El funcionamiento de Gades y Pegaso usando Java SE 6 es aparentemente correcto, es decir, hasta la fecha no se ha detectado ningún fallo durante la ejecución de Gades o Pegaso en el servidor.

Sin embargo, tras instalar **Java SE 7** en la máquina que aloja el servidor web, se ha comprobado que el funcionamiento de Gades y Pegaso no es correcto. Concretamente, se produce un error en tiempo de ejecución al hacer uso de la funcionalidad de evaluación del lenguaje de los sistemas. Una vez introducidas las respuestas a las preguntas realizadas por Gades o Pegaso, éstos las procesan y, entre otras acciones, operan sobre la BC con el fin de obtener un resultado para la evaluación en curso; es en ese momento en el que se produce un error que aborta el funcionamiento normal del sistema.

3.1.3. Descripción del error

El aviso acerca del error nos lo da, en primera instancia, un mensaje de error mostrado en pantalla, que se puede ver en la figura 22. Este mensaje aparece al llevar a cabo una evaluación del lenguaje que requiere la realización de una inferencia. Es decir, se produce cuando la decisión del sistema es una alarma (cuando la decisión del sistema es que el

[5] **Depurar** es el proceso consistente en identificar y corregir los errores de programación [WIK7].

resultado normal o es un aviso, el motor de inferencias no ha entrado en funcionamiento, y ha sido el propio código de la lógica de negocio el que ha determinado la decisión del sistema).

Figura 22. *Error producido al realizar una evaluación del lenguaje en Gades*

No obstante, para conocer exactamente dónde se produce el error, hemos de analizar la traza de la **excepción**. Las excepciones son el mecanismo usado en Java para gestionar errores y otros eventos excepcionales. Una excepción suele llevar un mensaje asociado para una mejor comprensión de la misma. Parte de la traza de la excepción obtenida en nuestro caso es la mostrada en figura 23 (observe que a la izquierda de cada figura de este tipo aparecen los números de cada línea; haremos referencia a ellos en algunas explicaciones).

```
1  java.lang.IllegalArgumentException: Comparison method violates its
2  general contract!
3     at java.util.TimSort.mergeLo(TimSort.java:747)
4     at java.util.TimSort.mergeAt(TimSort.java:483)
5     at java.util.TimSort.mergeForceCollapse(TimSort.java:426)
6     at java.util.TimSort.sort(TimSort.java:223)
7           at java.util.Arrays.sort(Arrays.java:727)
```

Figura 23. *Fragmento de la traza de la excepción producida al realizar una evaluación*

A raíz de esta excepción, que no se muestra completa por simplicidad, hemos detectado que el error se encuentra en la línea 1 del fragmento de código mostrado en la figura 24. Como el método usado, *listStatements*, pertenece a la clase *Model* de la librería *RDF* de Apache Jena, podemos afirmar que el error no tiene su origen en el código de las aplicaciones Gades y Pegaso, sino en dicho método.

```
1  StmtIterator i=m.listStatements(s,p,o);
2        for (; i.hasNext(); ){
3              Statement stmt= i.nextStatement();
```

Figura 24. *Fragmento de código de la clase* EvaluacionLenguajeController

A pesar de que, como se explica en [ORA14], Java SE 7 es altamente compatible con previas versiones de la plataforma Java y, por ello, casi todos los programas que funcionaban con Java SE 6 deberían funcionar con Java SE 7 sin realizar modificación alguna, hay algunas incompatibilidades menores que suponen raras circunstancias y casos aislados documentados en [ORA14]. Precisamente, uno de dichos "casos aislados" es el causante del problema de compatibilidad de Gades y Pegaso con Java SE 7. La descripción resumida de este problema de compatibilidad aparece en la figura 25.

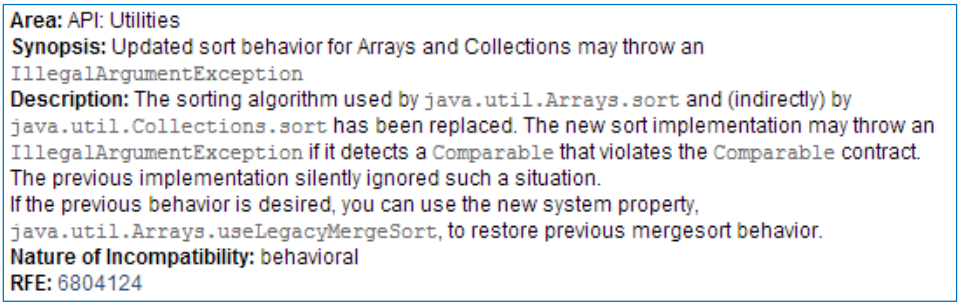
Area: API: Utilities
Synopsis: Updated sort behavior for Arrays and Collections may throw an `IllegalArgumentException`
Description: The sorting algorithm used by `java.util.Arrays.sort` and (indirectly) by `java.util.Collections.sort` has been replaced. The new sort implementation may throw an `IllegalArgumentException` if it detects a `Comparable` that violates the `Comparable` contract. The previous implementation silently ignored such a situation.
If the previous behavior is desired, you can use the new system property, `java.util.Arrays.useLegacyMergeSort`, to restore previous mergesort behavior.
Nature of Incompatibility: behavioral
RFE: 6804124

Figura 25. *Reporte del error en la página de Oracle [ORA14]*

Como en se muestra en la figura 25, este problema de compatibilidad se cataloga como una **incompatibilidad de comportamiento** (*Nature of Incompatibility: behavioral).* La compatibilidad de comportamiento existe cuando, con las mismas entradas, un programa realiza las mismas operaciones (o equivalentes) bajo diferentes versiones de las librerías de la plataforma Java. Hay que considerar que hay aspectos del comportamiento de la plataforma Java que, intencionalmente, no son especificados y la implementación que subyace puede cambiar de una plataforma a otra. Por ello, se recomienda que el código sea escrito de tal modo que no dependa de comportamientos sin especificar. En este escenario, el problema no es una incompatibilidad, sino un *bug* en el código [ORA14].

Por ejemplo, un método puede servir para realizar la misma operación en Java SE 6 y en Java SE 7, pero su implementación puede variar de una versión a otra. Este es el caso que nos atañe, pues, de acuerdo a lo expuesto en la figura 25, los métodos *java.util.Arrays.sort* y *java.util.Collections.sort*, que siguen presentes en Java SE 7, han variado su implementación de Java SE 6 a Java SE 7. La variación consiste en un cambio del algoritmo de ordenación de referencias a objetos usado por dichos métodos: se ha pasado del algoritmo *MergeSort* en Java SE 6 al algoritmo *TimSort* en Java SE 7. La razón del cambio es la mayor velocidad de este último [ORA13].

Como se puede ver en la figura 25, el algoritmo *TimSort* no ignora una situación que sí era ignorada por el algoritmo *MergeSort*: la violación del contrato *Comparable* en alguna parte del código. Este contrato especifica una serie de reglas acerca de cómo deben

compararse objetos entre sí, con el fin de ordenarlos. De este modo, el problema que ha aparecido en nuestros sistemas es un *bug* en el método de la librería *RDF* de Apache Jena antes mencionado, o en alguno de los métodos que este método pueda usar. Es decir, algún método de las librerías de Apache Jena viola el contrato *Comparable* y la nueva implementación del método *java.util.Arrays.sort* en Java SE 7 lo ha detectado

A pesar de haber buscado dentro del fichero *jena.jar* (contiene las librerías de Apache Jena) la versión de Jena usada, no hemos podido encontrarla. Esta es la razón por la que no hemos consultado el código fuente de la librería para encontrar el método exacto en el que se produce el error.

3.1.4. Solución escogida

En primer lugar, se intentó sustituir el fichero *jena.jar*, que contiene las librerías de Apache Jena usadas actualmente en el servidor web, por la última versión de las librerías de Apache Jena, con el objetivo de comprobar si se ha solucionado este error en las últimas versiones. Por las consultas realizadas acerca de las mejoras introducidas por cada nueva versión de Apache Jena, se sospecha que este error no ha sido resuelto. Aunque no conocemos su versión con exactitud, la fecha del archivo *jena.jar* ha sido una referencia aproximada para realizar dicha consulta.

Sin embargo, al sustituir el archivo *jena.jar* por la versión más reciente de las librerías de Apache Jena, el funcionamiento de Pegaso y Gades se veía afectado: aparecieron errores en la capa de lógica de las aplicaciones, concretamente en la parte en la que se lleva a cabo el proceso de inferencia, que no existían anteriormente. Por simplicidad y por mantener código en el que se realiza la inferencia cómo estaba, se decidió optar por mantener la versión de Jena en uso.

La solución por la que finalmente hemos optado, a raíz de lo explicado en la figura 25, es restablecer el antiguo comportamiento de los métodos *java.util.Arrays.sort* y *java.util.Collections.sort*, es decir, usar el antiguo algoritmo de ordenación, *MergeSort*, en lugar del nuevo, *TimSort*. En la práctica, la diferencia entre el uso de uno y otro algoritmo será imperceptible por el usuario final, por lo que no supone un inconveniente.

Para poner en práctica la solución escogida, es necesario activar la siguiente propiedad en el servidor web Apache Tomcat:

- **java.util.Arrays.useLegacyMergeSort**: esta propiedad será cargada por la JVM instalada en la máquina que aloja el servidor web.

El servidor Apache Tomcat tiene dos variables de entorno que nos permiten modificar la propiedad anterior [TOM7]:

- **CATALINA_OPTS** (*Catalina options*, donde Catalina es el nombre del contenedor de Servlets del servidor Apache Tomcat). Es una variable usada frecuentemente que permite especificar opciones adicionales para el comando Java que arranca Tomcat.
- **JAVA_OPTS** (*Java options*). Es una variable usada menos frecuentemente que permite especificar opciones que son usadas tanto en el arranque como en la parada de Tomcat.

Hemos elegido usar la primera variable de entorno, CATALINA_OPTS, por las siguientes razones [WEL09]:

- No es necesario que la propiedad *java.util.Arrays.useLegacyMergeSort* esté activada durante la parada del servidor, aunque en la práctica esto no es determinante.
- La variable CATALINA_OPTS es usada solamente por Apache Tomcat. Así, si en algún momento, en la máquina que aloja al servidor se instala otro servicio que nada tenga que ver con Apache Tomcat, este servicio no tendría por qué cargar dicha variable de entorno, como sí sucedería si usáramos la variable JAVA_OPTS.

Como la propia *Apache Software Foundation* recomienda [TOM7], se ha creado un *script*[6] llamado *setenv.bat* (*setenv* viene de *set environment variable*) para modificar la variable de entorno CATALINA_OPTS y activar la propiedad *java.util.Arrays.useLegacyMergeSort*. Parte del contenido de este *script* se muestra en la figura 26 (el resto del contenido son comentarios).

```
1  set CATALINA_OPTS=-Djava.util.Arrays.useLegacyMergeSort=true
```

Figura 26. *Contenido del* script setenv.bat

Este *script* no está presente por defecto en el servidor Apache Tomcat, así que lo ubicamos en la carpeta *~/apache-tomcat-7.0.50/bin/* del mismo (7.0.50 es la versión de Apache Tomcat usado durante el desarrollo de este PFG). En esa carpeta se encuentra, por defecto, el *script catalina.bat*, que entre otras acciones ejecuta el *script setenv.bat* en el momento en el que el servidor es arrancado. De este modo, se activa la propiedad

[6] Un ***script*** es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano. El uso habitual de los *scripts* es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario [WIK8].

java.util.Arrays.useLegacyMergeSort, obligando a la JVM de Java SE 7 a usar el algoritmo *MergeSort* que se usaba en Java SE 6 para las aplicaciones contenidas en el servidor Apache Tomcat.

En definitiva, el servidor web Apache Tomcat puede usar Java SE 7 renunciando simplemente al uso de un nuevo algoritmo de ordenación, dado que el resto de funcionalidades de Java SE 7 pueden ser empleadas. Podemos considerar, pues, que el problema de compatibilidad de Gades y Pegaso con Java SE 7 ha sido resuelto.

3.2. Generación de ontologías desde código Java

Comenzaremos estudiando la problemática de la generación de ontologías para los SE Pegaso y Gades (apartado 3.2.1). Tras ello, estudiaremos:

- El concepto de Web Semántica (apartado 3.2.2).
- El lenguaje OWL (apartado 3.2.3) y el formato de una ontología usando OWL DL (apartado 3.2.4).
- Las alternativas disponibles para generar una ontología OWL desde una aplicación Java (apartado 3.2.5).

Por último, se expresa y justifica la elección de un API que permita generar una ontología OWL DL desde código Java, y se explica cómo se ha construido la aplicación desarrollada a tal efecto (apartado 3.26).

3.2.1. Descripción de la necesidad

Hasta el comienzo de este PFG, la manera de generar una ontología para los SE Pegaso y Gades ha sido la utilización del programa Protégé. El uso de dicho programa requiere de una persona que sepa usarlo para actualizar las ontologías usadas en Pegaso y Gades.

Con objeto de independizar el proceso de generación de ontologías de los desarrolladores y administradores de las plataformas Pegaso y Gades, se desea desarrollar una aplicación Java capaz de generar una ontología en lenguaje OWL. Para ello, es necesario emplear un API externo que permita trabajar con el lenguaje OWL.

Por tanto, ha sido necesario llevar a cabo una investigación de las diferentes alternativas existentes que permiten realizar dicha tarea. Esta labor de investigación y elección de un API, así como el desarrollo de una primera versión de la aplicación usando este API, ha sido llevada a cabo de manera conjunta con Miguel Menéndez Álvarez, como también se explica

en su Proyecto Fin de Grado [MEN14]. La aplicación desarrollada permite generar una pequeña ontología válida en lenguaje OWL DL.

Esta tarea, que supone la **automatización del proceso de generación de ontologías** (sin intervención de personas con conocimientos de ontologías o instruidas en el uso de editores de ontologías como Protégé), ha sido continuada por Miguel Menéndez Álvarez en solitario y está debidamente documentada en [MEN14]. Como parte de dicho PFG, se ha generado una aplicación basada en la aplicación desarrollada conjuntamente que permite generar una ontología completa para los sistemas Pegaso y Gades.

El objetivo final es integrar dicha aplicación tanto en Pegaso como en Gades, de modo que los usuarios autorizados (profesionales de AP y profesionales de colegios infantiles, respectivamente) puedan realizar modificaciones sobre la ontología en uso y generar nuevas versiones de la misma de manera independiente.

3.2.2. Web Semántica

La **Web Semántica** (*Semantic Web*) es un movimiento colaborativo guiado por el *World Wide Web Consortium* (W3C) (una organización internacional de estándares para la web). Este estándar promueve formatos comunes de datos en la *World Wide Web* (la web).

La Web Semántica pretende convertir la web actual (dominada por documentos no estructurados y semiestructurados) en una "red de información", fomentando la inclusión de contenido semántico en las páginas web. En otras palabras, se pretende dar a la información un significado explícito, haciendo que sea más fácil para las máquinas procesar e integrar automáticamente información disponible en la web [W3C04].

La pila de la Web Semántica se basa en la familia de especificaciones RDF (*Resource Description Framework*) del W3C. OWL es parte de la pila de recomendaciones del W3C relacionadas con la Web Semántica [W3C04], como se puede ver en la figura 27.

Pila de recomendaciones
OWL
RDF Schema
RDF
XML y tipos de datos XML Schema

Figura 27. *Pila de recomendaciones definida por el W3C*

A continuación, se explican en mayor detalle las recomendaciones recogidas en la figura 27:

- **XML**. Proporciona una sintaxis superficial para documentos estructurados, pero no impone restricciones semánticas sobre el significado de estos documentos.
- **XML Schema**. Es un lenguaje para restringir la estructura de los documentos XML y además extiende XML con tipos de datos (*datatypes*).
- **RDF**. Es un modelo de datos para objetos ("recursos") y las relaciones entre ellos. Proporciona una semántica simple basada en la sintaxis de XML.
- **RDF Schema (RDFS)**. Proporciona un vocabulario de modelado de datos para describir propiedades y clases de recursos RDF. Es una extensión del vocabulario RDF básico.
- **OWL**. Añade más vocabulario para describir propiedades y clases. Es un lenguaje destinado a ser utilizado cuando la información contenida en documentos necesita ser procesada por aplicaciones, en oposición a situaciones cuando el contenido solamente necesita ser presentado a humanos.

3.2.3. Lenguaje de Ontologías Web (OWL)

OWL puede ser usado para representar explícitamente el significado de términos en **vocabularios** y las **relaciones entre esos términos**. Esta representación de términos y sus relaciones se denomina ontología. OWL tiene más facilidades para expresar significado y semántica que XML, RDF y RDFS. Por tanto, OWL va más allá que estos lenguajes en su habilidad para representar contenido interpretable por máquinas en la web. OWL es una revisión del lenguaje de ontologías web DAML+OIL que incorpora lecciones aprendidas a partir del diseño y aplicación de este [W3C04].

OWL provee tres lenguajes, cada uno con nivel de expresividad mayor que el anterior [W3C04]:

- **OWL Lite** ofrece soporte a aquellos usuarios que principalmente necesitan una clasificación jerárquica y restricciones simples. OWL Lite tiene una menor complejidad formal que OWL DL.
- **OWL DL** ofrece soporte a aquellos usuarios que desean la máxima expresividad conservando completitud computacional (se garantiza que todas las inferencias sean computables) y decibilidad (se garantiza que todos los cálculos terminaran en un

tiempo finito). OWL DL es nombrado de este modo debido a la correspondencia con lógicas descriptivas (DL, *Description Logic*), un área de investigación que ha estudiado la lógica que constituye la base formal de OWL.

- **OWL Full** está pensado para usuarios que quieren la máxima expresividad y la libertad sintáctica de RDF, aunque no ofrece garantías computacionales.

OWL Full es la versión más completa de las tres, OWL DL es algo más limitada y, a su vez, OWL Lite es más limitada aún que OWL DL. Se podría decir que OWL Full es una extensión de OWL DL, y que éste es una extensión de OWL Lite. Respecto a la compatibilidad, una ontología realizada en OWL Lite es correcta en OWL DL, pero esto no ocurre a la inversa [VAD07].

En Gades y Pegaso se usa OWL DL como lenguaje de implementación de la ontología.

3.2.4. Formato de una ontología OWL DL

El documento que contiene a cualquier ontología OWL, y en particular a las ontologías OWL DL usadas en Pegaso y Gades, tiene el formato mostrado en la figura 28.

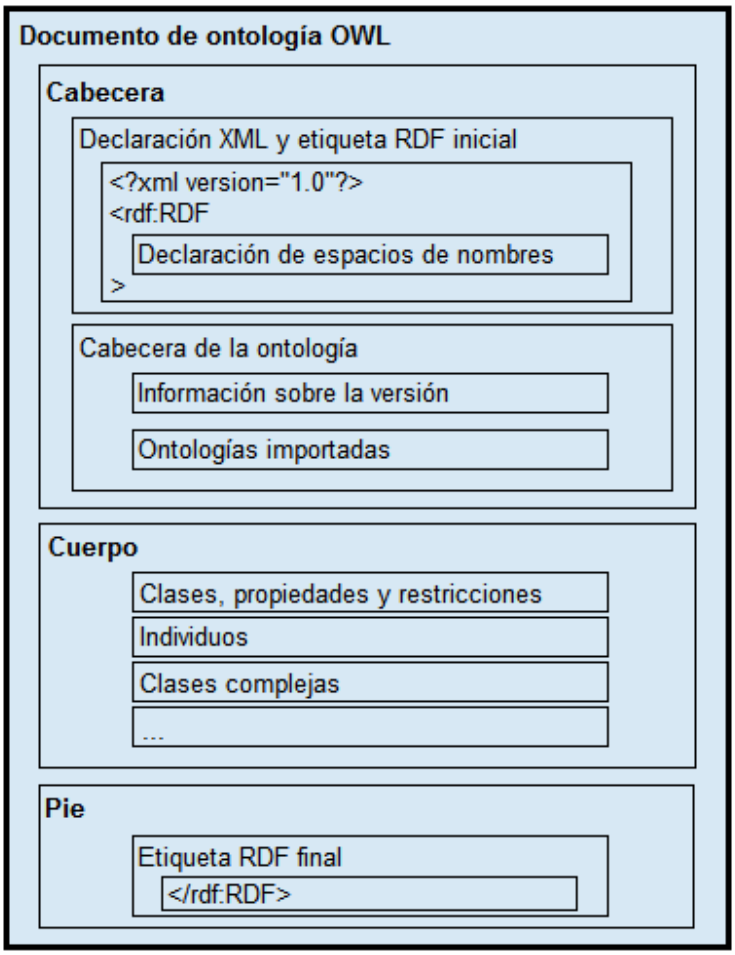

Figura 28. *Formato de una ontología OWL (elaboración propia a partir de [STR12])*

En la **cabecera** del documento se incluye una declaración de los espacios de nombres, justo después de la declaración de la versión de XML usada y la etiqueta de apertura inicial de RDF. En la cabecera también se puede incluir otra información, como la versión de la ontología, y se pueden importar otras ontologías para ser utilizadas por esta.

El **cuerpo** del documento contiene las clases, propiedades e individuos de la ontología, entre otros elementos.

Por último, el **pie** del documento simplemente contiene la etiqueta RDF de cierre.

Los componentes contenidos en la ontología OWL DL son explicados en mayor detalle a continuación, apoyando las explicaciones con ejemplos extraídos de una de las últimas versiones de la ontología usada en Pegaso.

Espacios de nombres

Antes de poder usar un conjunto de términos, necesitamos una indicación precisa de qué **vocabularios específicos** están siendo usados. El componente inicial estándar de una ontología incluye un conjunto de declaraciones de espacios de nombres XML (*XML namespaces*) encerradas en una etiqueta de apertura *rdf:RDF*. Estos espacios de nombres proveen un medio para interpretar inequívocamente identificadores y hacer el resto de la ontología mucho más legible. Una ontología OWL típica comienza con una declaración de espacios de nombres similar a la mostrada en la figura 29 [W3C04].

```
1  <rdf:RDF
2      xmlns="http://www.owl-ontologies.com/Ontology1302172874.owl#"
3      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4      xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
5      xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
6      xmlns:owl="http://www.w3.org/2002/07/owl#"
7      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
8      xmlns:swrl="http://www.w3.org/2003/11/swrl#"
9      xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
10     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
11 xml:base="http://www.owl-ontologies.com/Ontology1302172874.owl">
```

Figura 29. *Espacios de nombres la ontología de ejemplo*

Por ejemplo, *xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"* (línea 3) identifica al vocabulario RDF, siendo "http://www.w3.org/1999/02/22-rdf-syntax-ns" el identificador, o URI (*Uniform Resource Identifier*), del espacio de nombres de RDF.

La ontología en sí constituye un nuevo espacio de nombres, cuyo URI es el asignado al atributo *xmlns* (línea 2): "http://www.owl-ontologies.com/Ontology1302172874.owl".

Cabecera de la ontología

Una vez establecidos los espacios de nombres, aparece la cabecera, en la que se incluyen una colección de aserciones acerca de la ontología agrupadas bajo la etiqueta *owl:Ontology*. Estas aserciones pueden ser: comentarios, información para el control de versiones o la inclusión de otras ontologías. El elemento *owl:Ontology* es el lugar donde se pueden recoger muchos de los metadatos del documento [W3C04].

En nuestro caso, no se recoge ninguna información en la cabecera, como podemos observar en la figura 30.

```
1  <owl:Ontology rdf:about=""/>
```

Figura 30. *Cabecera de la ontología de ejemplo*

El atributo *rdf:about* provee un nombre o referencia para la ontología. Como el valor, en la ontología analizada, es "", el nombre de la ontología es el dado por el atributo *xml:base* y coincide con el URI de la ontología ("http://www.owl-ontologies.com/Ontology1302172874.owl").

Por otra parte, si desde una nueva ontología se decidiera usar la ontología de ejemplo, habría que importarla en la cabecera de la nueva ontología, quedando dicha cabecera como se muestra en la figura 31.

```
1  <owl:Ontology rdf:about="">
2      <owl:imports rdf:resource="http://www.owl-
3      ontologies.com/Ontology1302172874.owl"/>
4  </owl:Ontology>
```

Figura 31. *Cabecera de una nueva ontología si se importara la ontología de ejemplo*

Clases

Ya dentro del cuerpo de la ontología, aparecen las **clases**. Los conceptos más básicos en un dominio deben corresponder a clases que sean las raíces de varias taxonomías[7]. Cada individuo en el mundo OWL es miembro de la clase *owl:Thing*. Luego, cada clase definida por un usuario es implícitamente una subclase de *owl:Thing*. Las clases raíces específicas del dominio son definidas declarando simplemente su nombre. OWL también define la clase vacía, *owl:Nothing* [W3C04].

[7] Una **taxonomía** es un sistema de tipos jerárquicos que se puede utilizar para describir entidades. Los tipos se expresan en un sistema de clases y subclases [IBM12].

Las clases pueden contener **subclases**. Para indicar que una clase X es subclase de una clase Y, se usa el atributo *rdfs:subClassOf*. Esta relación es transitiva, es decir, si X es una subclase de Y e Y es una subclase de Z, entonces X es una subclase de Z.

En la ontología de ejemplo hay dos clases que son la raíz del resto. Como se puede ver en la figura 32, estas clases son:

- "AvanceSL" (línea 1). Esta clase contiene todos los hitos de desarrollo que deben ser alcanzados para considerar que el nivel de adquisición del lenguaje del niño es acorde a lo esperado. Estos hitos se traducen en las preguntas que el sistema realiza a los usuarios.
- "DecisionSistema" (línea 2). Esta clase contiene todas las decisiones que puede tomar el sistema.

```
1  <owl:Class rdf:ID="AvanceSL"/>
2  <owl:class rdf:ID="DecisionSistema"/>
```

Figura 32. *Clases raíz de la ontología de ejemplo*

La clase "AvanceSL" contiene las subclases que aparecen en la figura 33. De entre las subclases de "PrimerAño", tan sólo se muestran las subclases de la clase "Meses3" por simplicidad.

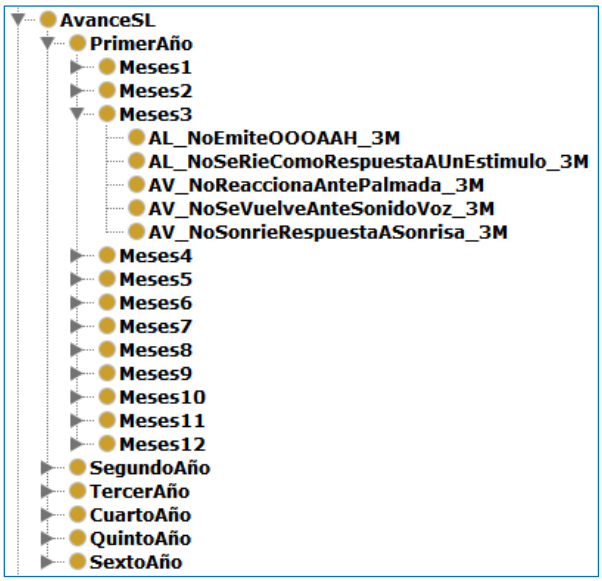

Figura 33. *Subclases de la clase "AvanceSL" (extraída de Protégé)*

Podemos ver que en el mes 3 hay cinco hitos de desarrollo. Uno de ellas es la clase "AL_NoEmiteOOOAAH_3M" ("AL" viene de alarma), que se corresponde con la pregunta "¿Emite OOO/AAH?", como podemos ver en el fragmento de la BC reflejado en la figura 34.

Hito	Descripción (pregunta que el usuario del sistema responderá con objeto de evaluar el estado de adquisición del lenguaje en el niño)	Decisión Sistema Neuropediatra
3 meses - Aviso	Se vuelve o reacciona (cerrando los ojos) ante una palmada	Adelantar visita (en tres meses)
3 meses - Aviso	Se vuelve a un sonido de voz madre	Adelantar visita (en tres meses)
3 meses - Aviso	Sonríe como respuesta a sonrisa	Adelantar visita (en tres meses)
3 meses - Alarma	Emite "OOO/AAH"	• Comprobar si está descartado problema de audición • Derivar al neuropediatra
3 meses - Alarma	Se ríe como respuesta a un estímulo	Derivar al Neuropediatra

Figura 34. *Fragmento de la BC correspondiente a los hitos de desarrollo del mes 3 [MAR11]*

Esta pregunta aparecerá en el documento OWL tal y como vemos en la figura 35. La clase "AL_NoEmiteOOOAAH_3M" es subclase de "Meses3" (ver líneas 7 a 9), "Meses3" es a su vez subclase de la clase "PrimerAño" (ver líneas 4 a 6) y ésta es a su vez subclase de "AvanceSL" (ver líneas 1 a 3).

```
1  <owl:Class rdf:about="#PrimerAño">
2      <rdfs:subClassOf rdf:resource="#AvanceSL"/>
3  </owl:Class>
4  <owl:Class rdf:ID="Meses3">
5      <rdfs:subClassOf rdf:resource="#PrimerAño"/>
6  </owl:Class>
7  <owl:Class rdf:about="#AL_NoEmiteOOOAAH_3M">
8      <rdfs:subClassOf rdf:resource="#Meses3"/>
9  </owl:Class>
```

Figura 35. *Algunas clases y subclases de la ontología de ejemplo*

Por otro lado, la clase "DecisionSistema" contiene las subclases que se pueden ver en la figura 36. Las decisiones del sistema se clasifican en dos tipos, según su gravedad: avisos (implican fijar próxima visita) y alarmas (implican derivar al especialista). De entre las subclases de "HitoAlarma" e "HitoAvisto", tan sólo se muestran las subclases de "Año1" por simplicidad.

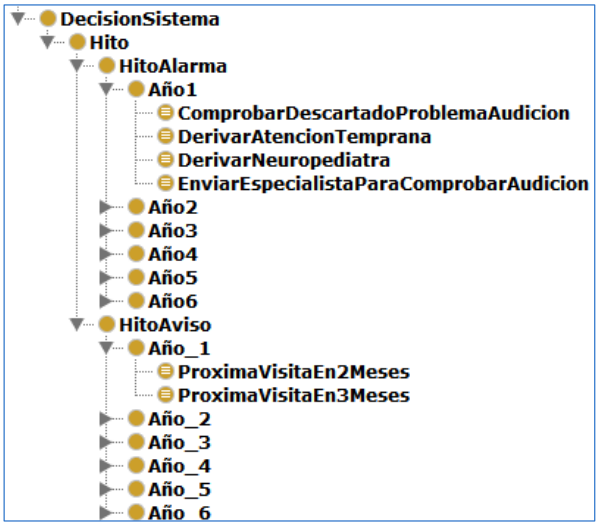

Figura 36. *Subclases de la clase "DecisionSistema" (extraída de Protégé)*

Las decisiones asociadas a la pregunta "AL_NoEmiteOOOAAH_3M" son "ComprobarDescartadoProblemaAudicion" y "DerivarNeuropediatra", que son de tipo alarma. Más adelante veremos que las preguntas se relacionan con las decisiones mediante propiedades.

Individuos

Además de clases, queremos ser capaces de describir a sus miembros. Normalmente pensamos en ellos como individuos (*individuals*) en este mundo de cosas (*things*). Un individuo se introduce simplemente al declararlo como miembro de una clase [W3C04]. Así, como vemos en la figura 37, "AV_I_NoEmiteOOOAAH_3M" (identificado por el atributo *rdf:ID*), es un individuo de la clase "AV_NoEmiteOOOAAH_3M".

```
1  <AL_NoEmiteOOOAAH_3M rdf:ID="AL_I_NoEmiteOOOAAH_3M"/>
```

Figura 37. *Individuo de la ontología de ejemplo*

Propiedades simples

Todo el conjunto de clases e individuos sería poco interesante si sólo pudiéramos definir taxonomías. Las propiedades nos permiten afirmar hechos generales acerca de los miembros de las clases y hechos específicos sobre los individuos [W3C04].

Se distinguen dos tipos de propiedades:

- **Propiedades de objetos** (*object properties*): establecen relaciones entre instancias de dos clases.
- **Propiedades de tipos de datos** (*datataype properties*): establecen relaciones entre clases y elementos RDF (estos elementos, o nodos, son conocidos como *literals* en RDF) o *datatypes* de XML Schema.

En la ontología de ejemplo solamente hay una propiedad: "hayRespuestaNegativaEn". Esta propiedad, que es de tipo *object property*, relaciona instancias de la clase "DecisionSistema" (*domain*) con instancias de la clase "AvanceSL" (*range*), como podemos ver en la figura 38.

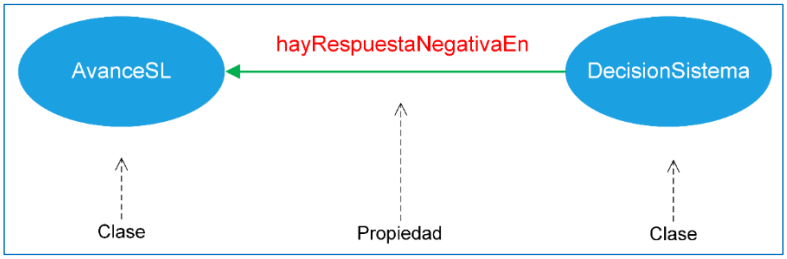

Figura 38. *Relación entre las clases "DecisionSistema" y "AvanceSL"*

Es decir, dicha propiedad apunta a las decisiones (pertenecientes a la clase "DecisionSistema") que le corresponden a cada hito de desarrollo (pertenecientes a la clase "AvanceSL").

En el documento OWL esta relación se materializa del modo que podemos ver en la figura 39.

```
<owl:ObjectProperty rdf:about="#hayRespuestaNegativaEn">
    <rdfs:range rdf:resource="#AvanceSL"/>
    <rdfs:domain rdf:resource="#DecisionSistema"/>
</owl:ObjectProperty>
```

Figura 39. *Propiedad en la ontología de ejemplo*

Restricciones de propiedades

Es posible restringir el rango (*range*) de una propiedad en contextos específicos. Para ello usamos restricciones de propiedades (*property restrictions*). El elemento *owl:onProperty* indica la propiedad restringida [W3C04].

En la ontología de ejemplo se usan las restricciones de propiedades siguientes:

- *allValuesFrom*: esta restricción requiere que para cada instancia de la clase que tiene instancias de la propiedad especificada, los valores a los que apunta la propiedad sean todos miembros de la clase indicada por la restricción *owl:allValuesFrom*.
- *someValuesFrom*: describe una clase de individuos para los cuales al menos un valor de la propiedad en cuestión es una instancia de la clase descrita por la restricción *owl:someValuesFrom*.

En la figura 40 podemos ver un ejemplo de la restricción ***allValuesFrom***.

```
<owl:Class rdf:about="#DecisionSistema">
    <rdfs:subClassOf>
     <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hayRespuestaNegativaEn"/>
      </owl:onProperty>
      <owl:allValuesFrom rdf:resource="#AvanceSL"/>
     </owl:Restriction>
</owl:Class>
```

Figura 40. *Restricción de propiedad* allValuesFrom *en la ontología de ejemplo*

En este ejemplo, para todos los individuos de la clase "DecisionSistema", la propiedad "hayRespuestaNegativaEn" apunta a un individuo de la clase "AvanceSL". Dicho de otro

modo, siempre que haya una decisión del sistema (individuo perteneciente a la clase "DecisionSistema"), se habrán producido una o varias respuestas negativas en las preguntas asociadas a uno o varios hitos (individuos de la clase "AvanceSL"). En caso contrario, es decir, si no hay respuestas negativas a las preguntas ("sí" o "no sabe/no contesta"), no existirá un individuo de la clase "DecisionSistema". Esto no quiere decir que el sistema no muestre una decisión por pantalla al usuario, pero no la habrá tomado el motor de inferencias, sino que estará predefinida en la lógica del programa Java.

Tenga en cuenta que por respuesta negativa se entiende responder "no" a una pregunta.

Por otra parte, podemos ver un ejemplo de la restricción ***someValuesFrom*** en la figura 41. Vemos que siempre que la decisión del sistema sea "DerivarLogopeda_Año6" tiene que haber al menos una respuesta negativa en un individuo de la clase "AL_NoPronunciaCorrectamenteFonemas_69M". Por ejemplo, si la decisión del sistema en una evaluación del lenguaje usando las preguntas correspondientes al mes 69 es "derivar al logopeda", ha habido, al menos, un "no" (una respuesta negativa) a la pregunta "¿el niño pronuncia correctamente los fonemas: 'r', 'bra', 'gra', 'tra, 'bla'?". Esto no implica que no haya habido más respuestas negativas a otras preguntas, pero lo que es seguro es que la respuesta a la citada pregunta ha sido "no". Para conocer el texto de la pregunta asociada al hito identificado por "AL_NoPronunciaCorrectamenteFonemas_69M" hemos consultado la BC.

```
1  <owl:Class rdf:ID="DerivarLogopeda_Año6">
2      <owl:equivalentClass>
3       <owl:Restriction>
4        <owl:onProperty>
5         <owl:ObjectProperty rdf:about="#hayRespuestaNegativaEn"/>
6        </owl:onProperty>
7         <owl:someValuesFrom
8         rdf:resource=="#AL_NoPronunciaCorrectamenteFonemas_69M"/>
9       </owl:Restriction>
10     </owl:equivalentClass>
11     <rdfs:subClassOf>
12      <owl:Class rdf:ID="Año6"/>
13     </rdfs:subClassOf>
14 </owl:Class>
```

Figura 41. *Restricción de propiedad* someValuesFrom *en la ontología de ejemplo*

Equivalencia entre clases y propiedades

En la ontología de ejemplo aparece el siguiente elemento para expresar equivalencia entre clases y propiedades:

- *equivalentClass*: se utiliza para indicar una condición necesaria y suficiente. Es decir, como vemos en la figura 41, para que la decisión del sistema sea "DerivarLogopeda_Año6", es necesario y suficiente que haya una respuesta negativa en la pregunta asociada al hito "AL_NoPronunciaCorrectamenteFonemas_69M". Dicho de otro modo, si la respuesta a la pregunta "¿el niño pronuncia correctamente los fonemas: 'r', 'bra', 'gra', 'tra, 'bla'?" es "no", entonces la decisión del sistema será "derivar el niño al logopeda".

Clases complejas

OWL provee constructores adicionales con los que formar clases. Uno de estos constructores es el siguiente:

- *unionOf*: se usa para crear una clase compuesta por varias clases.

Por ejemplo, en la figura 42 podemos ver que se define una clase que está formada por dos clases: "AV_NoComprende_Preguntas_Habituales_14M" y "AV_NoDiceAlgunaAccion_19M".

```
1 <owl:Class>
2     <owl:unionOf rdf:parseType="Collection">
3      <owl:Class
4      rdf:about="#AV_NoComprende_Preguntas_Habituales_14M"/>
5      <owl:Class rdf:about="#AV_NoDiceAlgunaAccion_19M"/>
6     </owl:unionOf>
7 </owl:Class>
```

Figura 42. *Clase compleja de la ontología de ejemplo creada usando* unionOf

3.2.5. Alternativas para generar una ontología OWL desde código Java

Existen varias alternativas para generar una ontología en lenguaje OWL desde código Java. Algunas de ellas son: Apache Jena, OWL API y Protégé-OWL. Estos API son descritos a continuación.

Apache Jena

Jena, inicialmente desarrollado por investigadores de HP Labs, y desde 2010 adoptado por la *Apache Software Foundation*, es un conjunto de herramientas entre las que se encuentran los siguientes API para Java [JEN14]:

- *RDF*: este es el API central de Apache Jena. Permite acceder, manipular y escribir información almacenada en gráficos RDF.

- *Ontology*: este API permite trabajar con modelos (*Model* es el contenedor principal de la información RDF), RDF Schema y OWL para añadir semántica adicional a los datos RDF.
- *Vocabulary*: este API contiene clases con constantes predefinidas para clases y propiedades definidas en vocabularios conocidos, como la clase *OWL:Thing*.

OWL API

OWL API, software libre y de código abierto, es un API para Java que permite crear, manipular y serializar[8] ontologías OWL. La última versión del API está enfocada hacia OWL 2. Este API es mantenido principalmente en la Universidad de Manchester (Reino Unido), pero ha habido importantes contribuciones de la compañía Clack & Parsia LLC y de la Universidad de Ulm (Alemania) [OWL14].

Protégé-OWL

Protégé-OWL, software libre y de código abierto, es un API para Java, que permite trabajar con los lenguajes OWL y RDF/RDF Schema. Este API provee clases y métodos para cargar y guardar archivos OWL, para consultar y manipular modelos de datos de OWL, y para llevar a cabo razonamientos basados en razonadores DL.

Este API está diseñado para ser usado en dos contextos [PRO10]:

- Para el desarrollo de componentes que son ejecutados en el interior de la interfaz de usuario del editor Protégé (este es el programa que se usó para desarrollar la ontología).
- Para el desarrollo de aplicaciones *stand-alone* (aplicaciones que se ejecutan localmente en una máquina).

API elegido para generar una ontología desde código Java

Hemos decidido descartar **Protégé-OWL** por estar más orientado a la implementación de aplicaciones que permitan el manejo de ontologías a través de interfaces gráficas de usuario que a la generación de ontologías desde código Java.

Por otra parte, consideramos que **OWL API** es una buena opción si se decidiera usar OWL 2 (última versión del lenguaje OWL), ya que las últimas versiones de este API se

[8] La **serialización** es el proceso de codificación de un objeto en un medio de almacenamiento (por ejemplo, un archivo o un buffer de memoria) [WIK9].

centran en OWL 2 nativamente. Pese a esto, se ha decidido escoger **Apache Jena** porque algunos de los servicios ofrecidos por Apache Jena ya se usaban en las aplicaciones Gades y Pegaso con anterioridad al comienzo de este PFG, como se explica en el apartado 2.2.2. Por ello, el grado de familiarización con él es mayor que con los otros API, con lo cual su uso resulta más intuitivo.

3.2.6. Primera versión de la aplicación Java desarrollada

Usando los API de Apache Jena, hemos desarrollado una aplicación que genera una pequeña ontología que contiene las preguntas asociadas a los hitos de desarrollo correspondientes al mes 1 (año 1) de la BC de Pegaso.

Como podemos ver en el fragmento de la BC mostrado en la figura 43, los hitos de desarrollo y la decisión del sistema correspondientes a dicho mes son los siguientes:

- Hitos:
    - "Reacciona a una campana": se corresponde con la clase "AL_NoReaccionaAUnaCampana" en la ontología.
    - "Vocaliza sin llorar": se corresponde con la clase "AL_NoVocalizaSinLlorar" en la ontología.

Hito	Descripción (pregunta que el usuario del sistema responderá con objeto de evaluar el estado de adquisición del lenguaje en el niño)	Decisión Sistema Neuropediatra
1 mes - Alarma	Reacciona a una campana	Enviar al especialista correspondiente para comprobar audición
1 mes - Alarma	Vocaliza sin llorar	Enviar al especialista correspondiente para comprobar audición

Figura 43. *Fragmento de la BC correspondiente a los hitos de desarrollo del mes 1 [MAR11]*

En la ontología OWL, estos hitos pertenecen a la clase "Meses1", que es subclase de "PrimerAño", la cual es a su vez subclase de "AvanceSL", como vemos en la figura 44. Estos hitos tienen asociada una decisión del sistema que es de tipo alarma, con lo cual el motor de inferencias inferirá usando la BC para obtener la decisión. De este modo, podremos poner a prueba la ontología que genere la aplicación desarrollada.

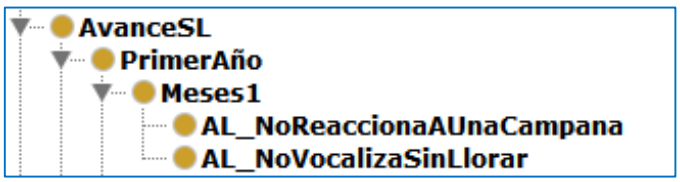

Figura 44. *Jerarquía de clases de los hitos de desarrollo correspondientes al mes 1*

- Decisión:
    - "Enviar al especialista correspondiente para comprobar audición": en la ontología OWL, esta decisión se corresponde con la clase "EnviarEspecialistaParaComprobarAudicion", que, como vemos en la figura 45, pertenece a la clase "Año1", que es subclase de "HitoAlarma" y, en última instancia, de la clase "DecisionSistema".

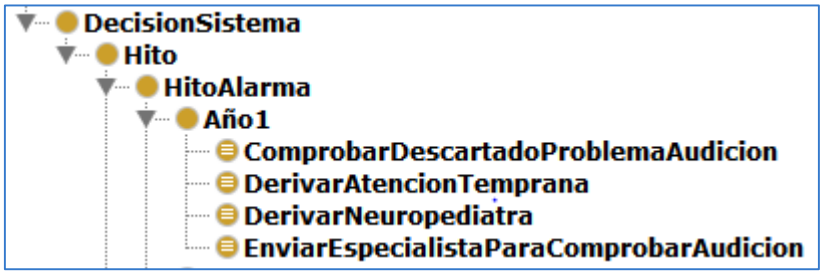

***Figura 45**. Jerarquía de clases de las decisiones del sistema correspondientes al año 1*

Considerando el formato de un **documento OWL** (ver apartado 3.2.4) y las clases que acabamos de describir (ver figuras 44 y 45), vamos a desarrollar una aplicación que permite generar una pequeña ontología para el mes 1.

El pseudocódigo[9] del método principal (*main*) de la aplicación desarrollada para es el mostrado en la figura 46 (en color verde aparecen los comentarios, es decir, no es necesario transformarlos en código Java). Los API de Apache Jena empleados para generar la ontología son los explicados en el apartado 3.2.5:

- *RDF*
- *Ontology*
- *Vocabulary*

[9] El **pseudocódigo** es una descripción de alto nivel compacta e informal del principio operativo de un programa informático u otro algoritmo [WIK10].

```
1  Inicio
2      //Crear ontología y cabecera
3      Crear ontología OWL DL
4      Añadir cabecera a la ontología
5      //Crear clases correspondientes a las preguntas
6      Crear clase “AvanceSL”
7      Crear clase “PrimerAño”
8      Crear clase “Meses1”
9      Crear clase “AL_NoReaccionaAUnaCampana”
10     Crear clase “AL_NoVocalizaSinLlorar”
11     Añadir “PrimerAño” como subclase de “AvanceSL”
12     Añadir “Meses1” como subclase de “PrimerAño”
13     Añadir “AL_NoReaccionaAUnaCampana” como subclase de “Meses1”
14     Añadir “AL_NoVocalizaSinLlorar” como subclase de “Meses1”
15     //Crear clases correspondientes a la decisión del sistema
16     Crear clase “DecisionSistema”
17     Crear clase “Hito”
18     Crear clase “HitoAlarma”
19     Crear clase “Año1”
20     Crear clase “EnviarEspecialistaParaComprobarAudicion”
21     Añadir “Hito” como subclase de “DecisionSistema”
22     Añadir “HitoAlarma” como subclase de “Hito”
23     Añadir “Año1” como subclase de “HitoAlarma”
24     Añadir “EnviarEspecialistaParaComprobarAudicion” como subclase de
25     “Año1”
26     //Crear los individuos
27     Crear individuo de la clase “AL_NoReaccionaAUnaCampana”
28     Crear individuo de la clase “AL_NoVocalizaSinLlorar”
29     Crear individuo de la clase “DecisionSistema”
30     //Crear la propiedad
31     Crear la propiedad “hayRespuestaNegativaEn”
32     Añadir rango a la propiedad
33     Añadir dominio a la propiedad
34     Añadir la propiedad al individuo de la clase “DecisionSistema”
35     //Crear las restricciones de la propiedad
36     Crear la restricción ‘allValuesFrom’
37     Añadir la restricción ‘allValuesFrom’ a la clase
38     DecisionSistema
39     Crear clase compleja formada por las dos preguntas creadas
40     Crear la restricción ‘someValuesFrom’
41     Crear equivalencia entre la clase “DecisionSistema” y entre la
42     restricción ‘someValuesFrom’ creada
43     //Guardar la ontología OWL
44     Escribir la ontología en un archivo de extensión .owl
45 Fin
```

Figura 46. *Pseudocódigo del método principal (*main*) de la aplicación desarrollada*

En el Anexo I, se puede consultar el fichero que contiene el código fuente de la aplicación desarrollada: *GeneradorOntologias.java*. La ontología OWL DL generada por esta aplicación, *ontologiaMes1.owl*, también se puede consultar en el dicho anexo.

Como explicamos en el apartado 3.2.1, a partir de esta primera versión de la aplicación desarrollada, Miguel Menéndez Álvarez ha generado una aplicación capaz de generar la ontología completa, como se explica en [MEN14].

3.3. Mejoras realizadas sobre las funcionalidades existentes en Pegaso y Gades

Se ha propuesto llevar a cabo varias mejoras sobre las funcionalidades existentes en Pegaso y Gades. Estas mejoras son las siguientes:

- En la funcionalidad de consulta de resultados, añadir la opción de imprimir un informe con los resultados de la consulta (apartado 3.3.1).
- Eliminar la dependencia con el URI de la ontología en el código de la clase *EvaluacionLenguajeController* (apartado 3.3.2).
- Cambiar el comportamiento de la lógica de negocio que transforma las respuestas "NS/NC" (no sabe/no contesta) en "NO" (apartado 3.3.3).
- Añadir la señal de *copyright* (©) junto con el nombre de la universidad en todas las páginas que se muestren en el navegador (apartado 3.3.4).

3.3.1. Creación de una opción para imprimir un informe de consulta de resultados

Descripción de la necesidad

Se desea añadir a la funcionalidad de consulta de resultados de las evaluaciones del lenguaje realizadas, en los sistemas Pegaso y Gades, la opción de imprimir un informe que contenga la misma información mostrada en la última pantalla de dicha funcionalidad, es decir:

- Datos generales del paciente: sexo, fecha de nacimiento, semanas de gestación, iniciales del nombre y peso al nacer.
- Resultado del proceso de evaluación del lenguaje:
    - Información sobre la propia evaluación: fecha de la evaluación, meses del niño y meses de las preguntas.
    - Preguntas realizadas durante la evaluación y respuestas introducidas.

- Respuesta, o respuestas, proporcionadas por el sistema.
- Indicación acerca de si el pediatra realizó o no lo propuesto por el sistema y, en caso negativo, explicación de por qué no realizó lo propuesto por el sistema.

- Valoración de la decisión del sistema del propio usuario que está consultando los resultados de una evaluación, si la ha valorado con anterioridad.

Diseño de la opción para imprimir

En primer lugar, vamos a presentar varios diagramas de comportamiento en lenguaje UML[10] (*Unified Modeling Language*). Un diagrama UML es una representación gráfica de una colección de elementos de modelado y sus relaciones. Nos permite representar el modelo de un sistema, esto es, una abstracción del sistema, o de una parte del sistema.

El **diagrama de casos de uso** de la figura 47 muestra los casos de uso asociados a la funcionalidad de consulta de resultados del sistema Pegaso. Este diagrama explica qué hace el sistema, sin especificar cómo lo hace. La funcionalidad de consulta de resultados la pueden usar los siguientes actores: el pediatra y el especialista médico.

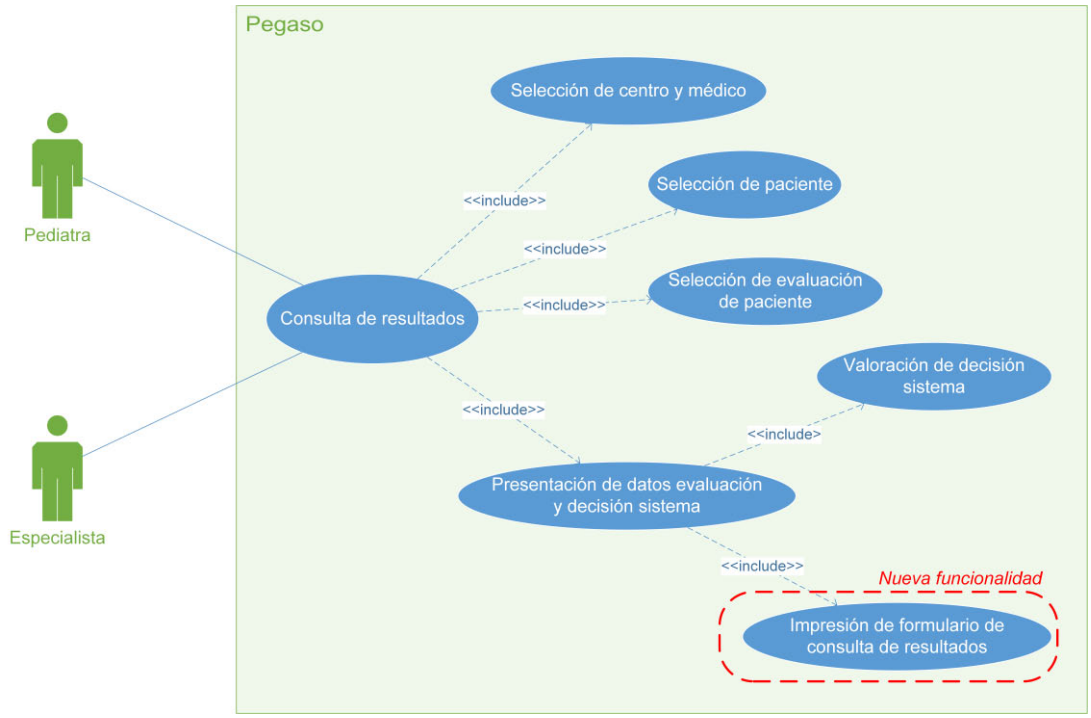

Figura 47. *Diagrama de casos de uso de la funcionalidad de consulta de resultados de Pegaso*

[10] **UML** es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema software orientado a objetos. No es un lenguaje de programación, sino un lenguaje casi formal (no tiene una semántica precisa para algunos elementos).

El caso de uso "consulta de resultados" incluye los siguientes casos de uso: selección de un centro y un médico, selección de un paciente, selección de una evaluación correspondiente al paciente seleccionado y presentación de los datos de la evalaución seleccionada y la decisión del sistema asociada a esta evaluación. Este último caso de uso incluye, a su vez, el caso de uso de valoración de la decisión del sistema y, la nueva funcionalidad (recuadrada en rojo), la impresión de un formulario de consulta de resultados.

El **diagrama de actividad** mostrado en la figura 48 presenta la funcionalidad asociada al caso de uso de consulta de resultados. En esta figura aparece un recuadro en color rojo con la nueva funcionalidad proporcionada por este PFG.

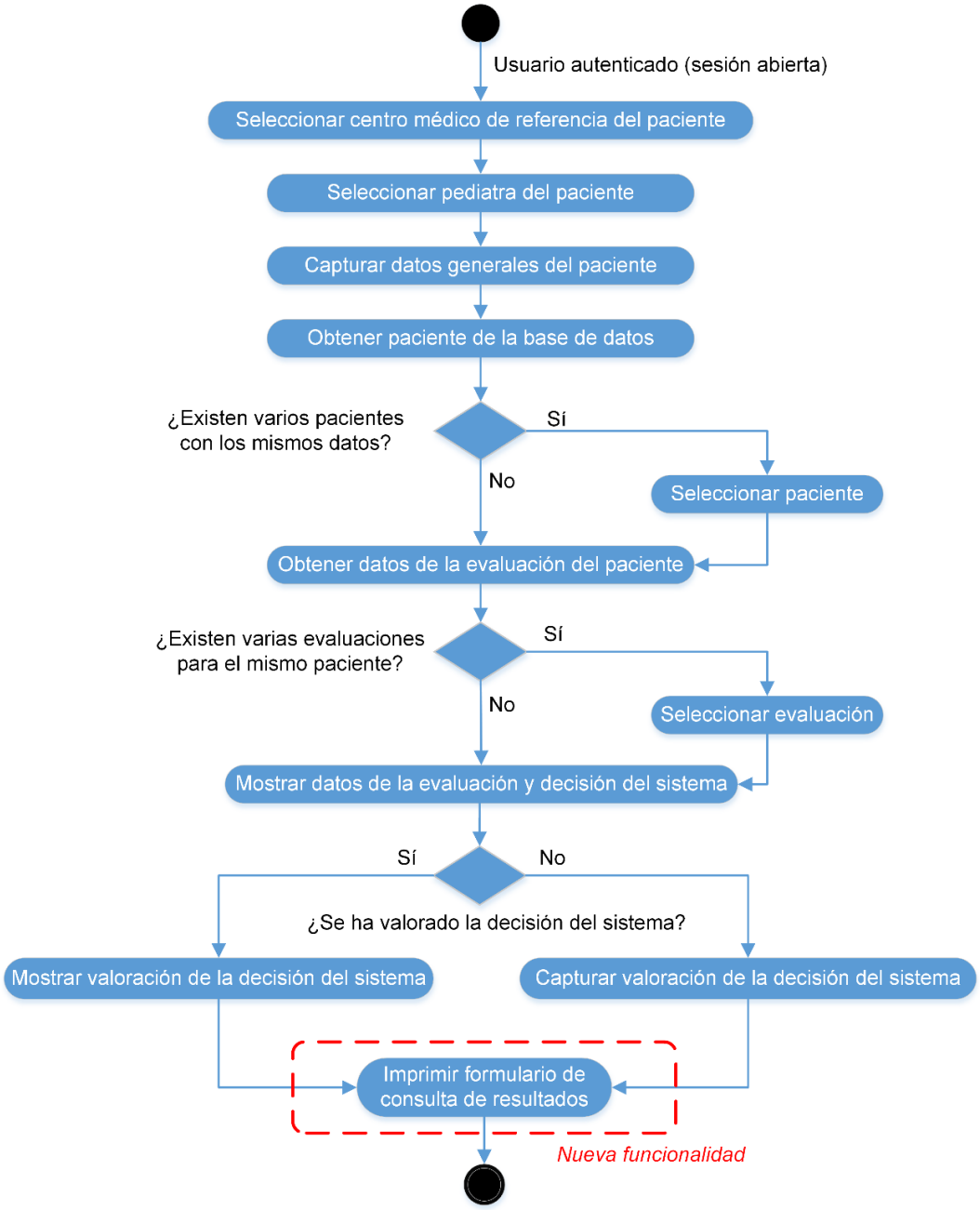

Figura 48. *Diagrama de actividad de la funcionalidad de consulta de resultados de Pegaso*

Implementación de la opción para imprimir

Una vez que hemos presentado el comportamiento de la funcionalidad de consulta de resultados, podemos comenzar a pensar en términos de qué tenemos que implementar en cada capa. Esto debe realizarse respetando la arquitectura del sistema (explicada en el apartado 2.2.2) y las funcionalidades desarrolladas hasta el momento, con objeto de hacer operativa la nueva funcionalidad.

El único nuevo elemento que ha sido necesario desarrollar es una nueva vista de usuario (JSP): *FormularioConsultaResultadosView*. El resto de la lógica necesaria ya está implementada en la funcionalidad de consulta de resultados.

En la vista de usuario en la que se muestra la evaluación del lenguaje consultada (consultar figura 20), *ValidacionResultadosView*, simplemente se ha añadido un botón "Imprimir" que permite abrir una ventana emergente que muestre la nueva vista (*FormularioConsultaResulatdosView*) con el informe de consulta de resultados listo para ser enviado a la impresora.

Es un problema que se limita, por tanto, a la capa de presentación. Basta con usar los atributos de sesión para obtener la información que hay que mostrar en el informe. Los atributos de sesión usados por la nueva opción son los siguientes:

Nombre de atributo	Tipo	Descripción
evaluacion	EvaluacionBean	Datos de la evaluación en la consulta de resultados
medico	MedicoBean	El médico que usa la aplicación
nombreCentro	String	Nombre del centro al que pertenece la evaluación que se está realizando o consultado
paciente	PacienteBean	El paciente con el que se está trabajando
preguntasRespuestas	HashMap	Mapa con las preguntas y respuestas correspondientes a una evaluación de un paciente
validacion	ValidacionBean	Validación de un médico a una determinada evaluación

Tabla 1. *Atributos de sesión usados para la impresión del informe de consulta de resultados*

El atributo "nombreCentro" se ha añadido en este proyecto, con objeto de poder resolver el problema; el resto ya estaban definidos en [URI13]. El atributo "nombreCentro" nos permite obtener el centro al que pertenece la evaluación consultada, mientras que del atributo "medico" obtenemos el nombre del médico que realizó la evaluación. Del atributo

"paciente" obtenemos los datos generales del paciente, de "evaluación" obtenemos la información sobre la propia evaluación, de "preguntasRespuestas" obtenemos las preguntas y sus respuestas. La decisión del sistema la obtenemos de "evaluación" y, por último, si existe, la validación se obtiene del atributo "validación".

Los valores de estos atributos de sesión han sido fijados durante el proceso de consulta de resultados por los diferentes ayudantes de vista que intervienen en el proceso. La interacción entre las vistas y los ayudantes de vista (capa de presentación) durante el proceso de consulta de resultados se muestra en la figura 49.

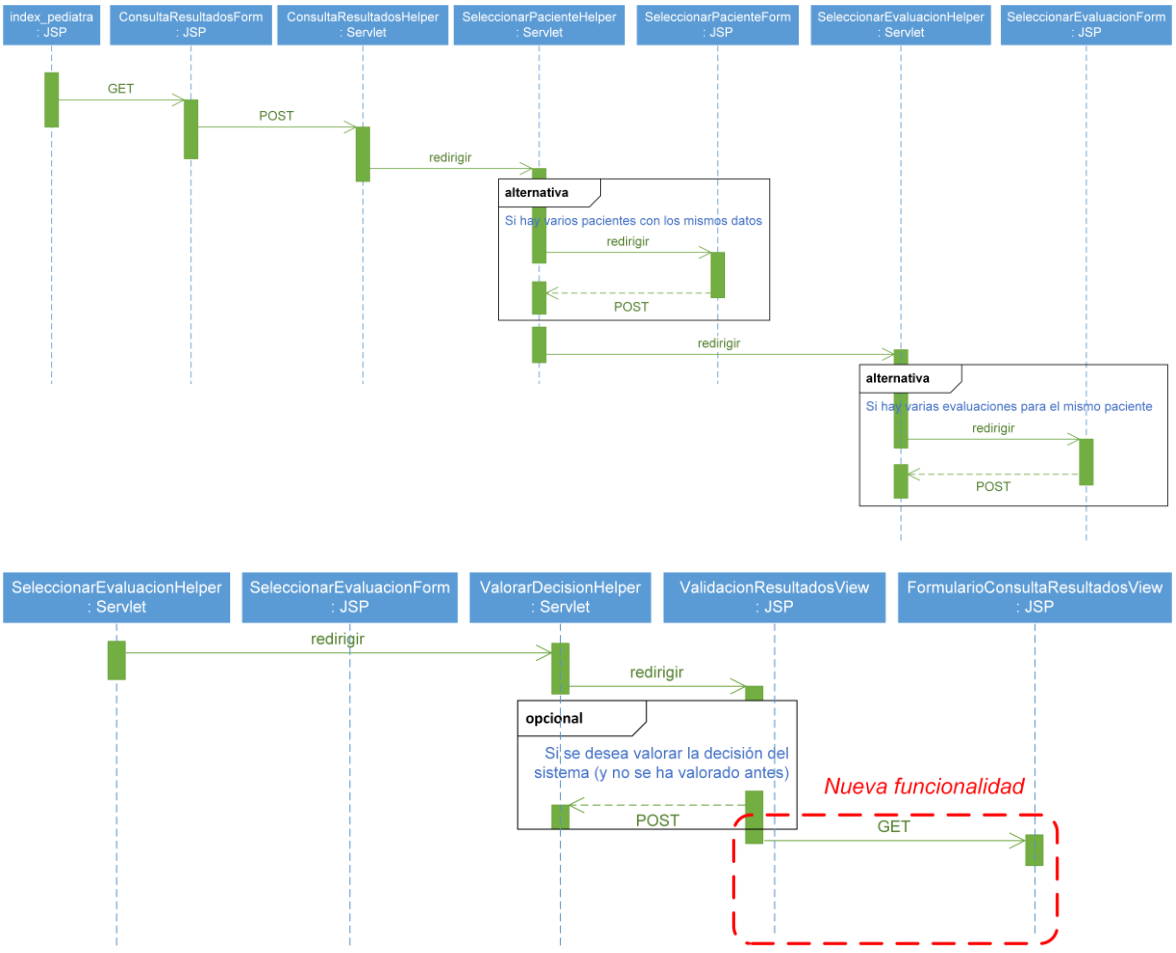

Figura 49. *Interacción en la capa de presentación de la consulta de resultados de Pegaso*

3.3.2. Dependencia con el URI de la ontología en la clase *EvaluacionLenguajeController*

Al tratar de generar una ontología desde código Java (apartado 3.2), se ha detectado que en la clase *EvaluacionLenguajeController* existe una dependencia con el URI (el identificador) que identifica al espacio de nombres (atributo *xmlns*) de las ontologías OWL usadas tanto en Pegaso como en Gades. Esto quiere decir que al cambiar la ontología usada

por otra con un URI diferente, ninguno de los dos sistemas es capaz de realizar un proceso de inferencia.

Siempre que se desarrolla una aplicación en cualquier lenguaje de programación es conveniente evitar dependencias con nombres o identificadores de archivos. Por ejemplo, antes que incluir el nombre de un fichero dentro de un método de una clase Java, es preferible usar un archivo de configuración externo para que la lógica de dicho método cargue del archivo de configuración el nombre del fichero. Esto se debe a que, en caso de que el nombre del fichero se modifique, es más sencillo modificar el archivo de configuración externo que buscar dentro del propio código Java el nombre del fichero, para actualizarlo.

En el caso que nos ocupa, la dependencia aparece en la línea 1 del fragmento de código de la clase *EvaluacionLenguajeController* que se muestra en la figura 50. El URI de la ontología en uso en los sistemas Pegaso y Gades, asignado por Protégé (ya que era la herramienta software usada para generar la ontología hasta el comienzo de este PFG), aparece de forma explícita en el código de la esta clase.

```
1  String URI= "http://www.owl-ontologies.com/Ontology1302172874.owl";
2  Resource resourceDecision= model.getResource(URI+"#DecisionSistema");
```

Figura 50. *Dependencia con el URI de la ontología en la clase* EvaluacionLenguajeController

Debido a esta dependencia, al sustituir la ontología antigua por la generada en el apartado 3.2.6, ni Pegaso ni Gades son capaces de inferir debido a que no encuentran los elementos dentro de la ontología, pues los buscan tomando como base ese URI, como podemos ver en la figura 50 (línea 2).

La solución se encuentra en los API de Apache Jena usados en la clase Java *EvaluacionLenguajeController* para acceder a la ontología. La clase *OntModel* posee un método denominado *getNsPrefixURI()* que permite obtener el URI de la ontología. Así, el nuevo código, que permite obtener el URI de manera dinámica directamente de la ontología, es el mostrado en la figura 51 (línea 1).

```
1  String URI= model.getNsPrefixURI("").split("#")[0];
2  Resource resourceDecision= model.getResource(URN+"#DecisionSistema");
```

Figura 51. *Solución para evitar la dependencia con el URI de la ontología*

3.3.3. Cambio en la lógica de negocio que transforma las respuestas "NS/NC" en "NO"

Tras mostrar la vista de usuario *PreguntasEvaluacionForm* (JSP) al cliente, donde éste responde a las preguntas planteadas durante el proceso de evaluación del lenguaje, el ayudante de vista *ResultadoConsultaHelper* transforma las respuestas "NS/NC" (no sabe/no contesta) en "no", en las preguntas correspondientes a los hitos de la clase "HitoAviso". Este es un comportamiento anómalo, ya que puede llegar a alterar la decisión del sistema tanto en Pegaso como en Gades. En la figura 52 podemos ver el fragmento del Servlet *ResultadoConsultaHelper* que da origen a esta anomalía (línea 3).

```
1  if(hito.getClass().getName().equals("data.HitoAvisoBean")
2  && respuestaParam.equals("NS/NC")){
3       respuestaParam = "No";
4  }
```

Figura 52. *Comportamiento no deseado en el ayudante de vista* ResultadoConsultaHelper

La solución a este problema pasa por eliminar las líneas de código anteriores. De este modo, no se altera el valor del atributo *respuestaParam* (de tipo *String*) cuando la respuesta es "NS/NC", y por tanto se almacena el valor correcto en la BD.

3.3.4. Adición del símbolo de *copyright* de la UPM

Con objeto de poder realizar el registro del software desarrollado se presenta la necesidad de añadir en todas las vistas (páginas JSP), así como en las páginas XHTML que corresponda, el símbolo de *copyright* seguido por el nombre de la universidad. Es decir, se desea añadir la siguiente línea al pie de cada una de estas páginas:

© Universidad Politécnica de Madrid

Para ello, simplemente se ha sustituido el texto que se mostraba anteriormente (© Departamento Diatel UPM) en todas las páginas JSP y XHTML por nuevo texto (© Universidad Politécnica de Madrid). El código que aparece al final de cada página JSP queda como se puede ver en la figura 53.

Observe que en la línea 4 se añade el símbolo de *copyright* y en las líneas 7 y 8 se añade el nombre de la universidad.

```
1  <table width="100%" border="0" align="center">
2    <tr>
3      <td align="center" class="Diatel">
4        <span class="Departamento_Diatel">&copy;</span>
```

```
5        <span class="Departamentp_Diatel"><span class="Diatel">
6          <a href="http://www.upm.es/institucional" title="Web
7          Institucional UPM" target="_new" class="Diatel">Universidad
8          Politécnica de Madrid</a>
9        </span></span>
10     </td>
11   </tr>
12 </table>
```

Figura 53. *Contenido común de las vistas (JSP) para mostrar la señal de* copyright

3.4. Desarrollo de una funcionalidad para explotación de datos en Pegaso y Gades

En este apartado se explica la funcionalidad que se ha desarrollado tanto en Gades como en Pegaso para la explotación de los datos (desde un punto de vista estadístico) con los que trabajan ambos sistemas, justificando su necesidad y describiendo detalladamente el diseño y la solución implementada.

3.4.1. Descripción de la necesidad

Una Base de Datos (BD) se define como "un **conjunto de datos** pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso" [WIK11]. Los sistemas Pegaso y Gades, como hemos explicado en apartados anteriores, cuentan con una BD gestionada por el gestor de BD MySQL. Cada BD almacena, de manera persistente, la información que cada sistema necesitará utilizar en un futuro, es decir, la información almacenada en la BD sobrevive tras el fin de la ejecución de la aplicación. Así, un posible corte en el suministro eléctrico, por ejemplo, en principio no afectaría a los datos ya almacenados en la BD, aunque el gestor de BD y/o el servidor dejaran de funcionar.

Los datos han sido almacenados en tablas, estando relacionadas unas con otras de acuerdo a una lógica definida en [URI13]. Sin embargo, estos datos por sí solos no constituyen información. La **información** se define como "un conjunto organizado de datos procesados que constituyen un mensaje que cambia el estado de conocimiento del sujeto o sistema que recibe dicho mensaje" [WIK12].

Por tanto, de no existir el proceso de consulta de resultados, no hablaríamos de información en los sistemas Pegaso y Gades, pues no existiría un mensaje claro, a simple vista, capaz de cambiar el estado de conocimiento del sujeto (el usuario) que observara dichos datos.

Ahora bien, la consulta de resultados solamente nos proporciona información de una evaluación de un paciente a la vez. De este modo, si se desea consultar información relativa a varias evaluaciones, de un mismo paciente o de diferentes pacientes, ha de realizarse un nuevo proceso de consulta de resultados. Es decir, si se desean consultar los resultados de diez evaluaciones del lenguaje, el usuario tiene que realizar diez procesos de consulta de resultados y, por tanto, tiene que introducir diez veces los datos de los pacientes, entre otras acciones. Si bien es posible, no resultaría nada práctico y el tiempo requerido sería en la mayoría de las ocasiones inasumible por un profesional médico o educativo.

Otra solución podría ser extender el proceso de consulta de resultados y mostrar en pantalla los datos de varias evaluaciones a la vez, como se hace para una sola evaluación. Sin embargo, no resultaría nada práctico; por el contrario, en muchos casos sería complicado procesar tal cantidad de información.

Descartadas las anteriores opciones, la opción de generar estadísticas a partir de los datos de las evaluaciones del lenguaje surge como una posibilidad ostensiblemente más sencilla y potente para analizar los datos de interés. Por tanto, se ha optado por desarrollar una nueva funcionalidad, separada de la consulta de resultados, que permita explotar estadísticamente la información referente a las evaluaciones almacenadas en los sistemas Pegaso y Gades.

3.4.2. Análisis de requisitos

En este apartado se pretende definir qué es lo que hace la funcionalidad de consulta de estadísticas. Para ello, vamos a definir los requisitos funcionales y no funcionales que deberá satisfacer la nueva funcionalidad.

Requisitos funcionales

Los **requisitos funcionales** establecen los comportamientos del sistema, es decir, los servicios que éste debe proporcionar, cómo debe reaccionar a una entrada particular y cómo debe comportarse en determinadas situaciones particulares.

En nuestro caso los requisitos funcionales se circunscriben al proceso de consulta de estadísticas y no al sistema completo. Los requisitos funcionales de la funcionalidad a desarrollar que describimos a continuación son válidos para Gades y Pegaso:

RF1. El proceso de consulta de estadísticas debe permitir a todos los usuarios autenticados en el sistema consultar estadísticas.

RF2. El proceso de consulta de estadísticas se realizará de forma interactiva. Se le pedirá al usuario información del centro (educativo o médico) en el que se realizaron

las evaluaciones, así como datos de los profesionales que realizaron las evaluaciones (puede haber varias combinaciones posibles).

RF3. Entre las opciones ofrecidas al usuario para seleccionar un centro y un profesional, se deben incluir opciones para seleccionar todos los profesionales de un centro y todos los profesionales de todos los centros.

RF4. El proceso de consulta de estadísticas debe permitir al usuario, tras seleccionar un centro y un profesional, la selección de una estadística de entre varias posibles, que estarán predefinidas. El texto de las diferentes estadísticas se redactará de manera clara (evitando imprecisiones y ambigüedades) y concisa (sin rodeos ni figuras retóricas).

RF5. Una vez que el usuario ha seleccionado un centro, un profesional y la estadística que desea consultar, la aplicación mostrará en pantalla una gráfica o una tabla que contenga los datos correspondientes a la estadística solicitada.

RF6. Se debe incluir una opción que permita imprimir la gráfica o la tabla mostrada en pantalla.

RF7. Si el usuario no selecciona una opción necesaria para avanzar en el proceso de consulta de estadísticas (véase centro, profesional y estadística deseada), se debe impedir su avance en el sistema y se debe mostrar un mensaje de error por pantalla que indique al usuario cómo debe proceder.

RF8. Si no existen evaluaciones del lenguaje realizadas por los profesionales seleccionados (pueden ser uno o varios), el proceso de consulta de estadísticas debe mostrar por pantalla un mensaje de error que informe al usuario de esta circunstancia y el proceso de consulta de estadísticas debe seguir funcionando correctamente.

RF9. Si la estadística seleccionada se muestra en una gráfica, ésta debe contener un título y una leyenda, así como los rótulos y/o las etiquetas necesarias para facilitar su entendimiento. Además, deberán emplearse colores para distinguir unas series de datos de otras. En la medida de lo posible, se evitarán los colores chillones, por ser desagradables para la vista del usuario.

RF10. Las tablas en las que se muestran estadísticas deben contener un título.

Requisitos no funcionales

Los **requisitos funcionales** afectan a los servicios o funciones del sistema, tales como restricciones sobre el proceso de desarrollo, restricciones de tiempo o estándares a utilizar, por ejemplo. Los requisitos no funcionales de la funcionalidad de consulta de estadísticas a desarrollar, aplicables tanto a Gades como a Pegaso, son los siguientes:

RNF1. El proceso de consulta de estadísticas ha de ser intuitivo, es decir, el usuario del sistema debe poder realizar un proceso de consulta de estadísticas completo (esto implica la generación de un elemento gráfica o tabla) en menos de 2 minutos.

RNF2. Las fuentes empleadas para el texto han de ser legibles (sin ribetes ni decoración excesiva) y, siempre que sea posible, se usarán los estilos ya definidos en la hoja de estilos CSS de la aplicación.

RNF3. El texto, las gráficas y las tablas que se muestran al usuario por pantalla deberán tener un tamaño tal que los usuarios sin problemas visuales puedan leerlos a una distancia, medida desde sus ojos a la pantalla, de 50 a 70 cm sin necesidad de forzar la vista o utilizar el *zoom*.

RNF4. La opción para imprimir la gráfica o la tabla mostrada en pantalla debe mostrarse en pantalla en un lugar accesible usando simplemente el ratón del ordenador y debe poder imprimirse el documento en menos de 5 clics con el ratón.

RNF5. Con objeto de salvaguardar el anonimato de los pacientes, no se podrán mostrar las iniciales del nombre y los apellidos de los pacientes. Sí se podrán mostrar las fechas de nacimiento de los pacientes.

RNF6. La funcionalidad de consulta de estadísticas ha de implementarse de tal modo que permita que el sistema siga siendo escalable y, para ello, ha de respetarse la arquitectura del sistema y han de usarse las tecnologías usadas hasta ahora.

RNF7. El API empleado para la generación de los elementos gráficos (gráficas o diagramas) ha de ser software libre y gratuito.

RNF8. Los componentes Java del proceso de consulta de estadísticas deben estar debidamente documentado para Javadoc[11].

RNF9. El proceso de consulta de estadísticas debe ser compatible con cualquiera de los navegadores web siguientes: Internet Explorer, Google Chrome y Mozilla Firefox. Es decir, el texto, los elementos gráficos y las tablas han de mostrarse correctamente en cualquiera de estos navegadores.

3.4.3. Diseño del módulo de consulta de estadísticas

Los diagramas UML son los mismos para Pegaso y Para Gades, por lo que en este apartado sólo se muestran los de Gades, por simplicidad. La úncia diferencia entre los

[11] **Javadoc** es una utilidad de Oracle para la generación de documentación de API en formato HTML a partir comentarios incluidos en el código fuente Java [WIK13].

diagramas de uno y otros sitema son los roles de los usuarios, conocidos como actores en UML, tal como se explicó en el apartado 1.1.

Modelo de la nueva funcionalidad

Por medio del **diagrama de casos de uso** mostrado en la figura 54 podemos ver qué hace la nueva funcionalidad, sin especificar cómo lo hace. Para el caso de uso de consulta de estadísticas, la nueva funcionalidad, se compone de varios casos de uso:

- Selección de centro educativo y profesional
- selección de la estadística deseada,
- presentación de la estadística solicitada, y, por último,
- posibilidad de imprimir en papel la información de la estadística consultada.

Los actores que pueden usar la nueva funcionalidad son todos los usuarios definidos para la plataforma Gades: educadores infantiles, logopedas y el administrador de la plataforma.

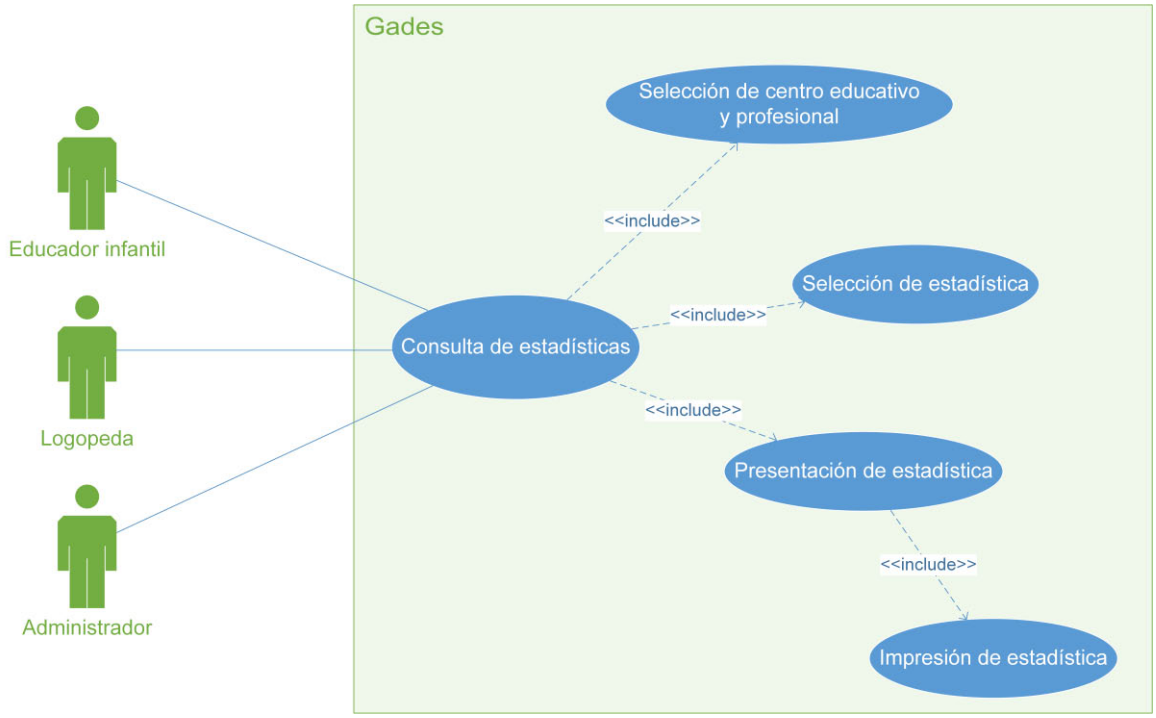

Figura 54. *Diagrama de casos de uso de la funcionalidad de consulta de estadísticas de Gades*

El **diagrama de actividad** mostrado en la figura 55 nos permite ver la secuencia de acciones que se lleva a cabo durante el proceso de consulta de estadísticas.

Una vez que el usuario se ha autenticado, elegirá un centro educativo y uno o varios profesionales. A continuación, el usuario elegirá qué estadística desea visualizar. Si estos profesionales han realizado evaluaciones del lenguaje, se mostrará la gráfica o tabla

correspondiente a la estadística seleccionada y, por último, el usuario contará con la opción de imprimir dicha gráfica o tabla.

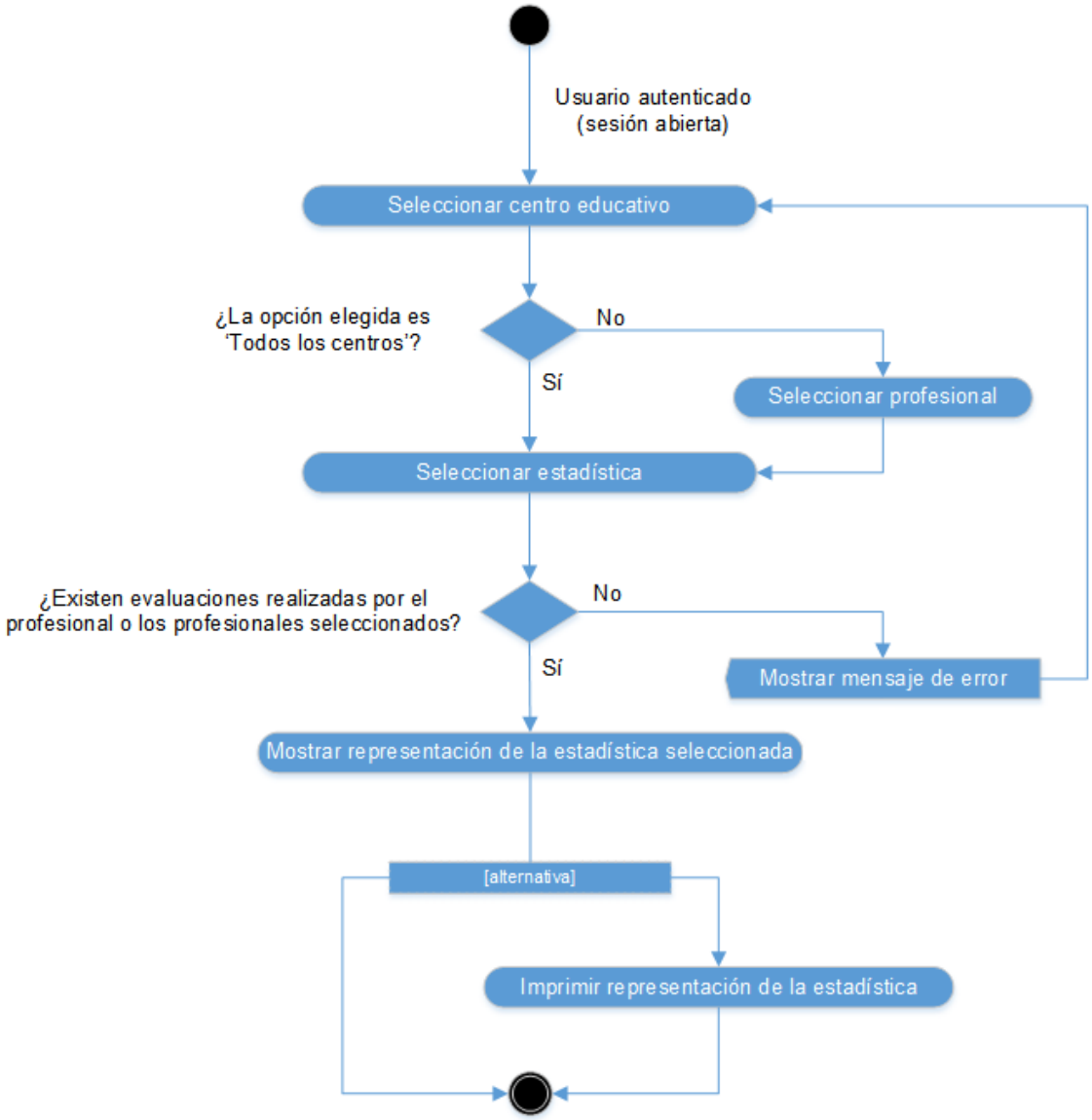

Figura 55. *Diagrama de actividad de la funcionalidad de consulta de estadísticas de Gades*

Estadísticas escogidas

Antes de pensar qué estadísticas queremos mostrar, vamos a ver con qué datos contamos. Las tablas almacenadas en la BD contienen los siguientes datos:

- **Datos de los usuarios para autenticación:**
  - Nombre de usuario
  - Contraseña.
- **Datos de los centros médicos** (Pegaso) **o educativos** (Gades):

  - Identificador del centro.
  - Nombre.
  - Dirección.
  - Localidad.
  - Provincia.
  - Comunidad autónoma.
  - Teléfono.
- **Datos de los usuarios** (profesionales médicos, en Pegaso; profesionales educativos, en Gades):
  - Identificador del usuario.
  - Nombre.
  - Apellidos.
  - Identificador del centro al que pertenece.
  - Identificador de su especialidad.
  - Número de colegiado (su valor es indiferente en Gades).
  - Estado del usuario en el sistema (activo o inactivo).
- **Datos de los pacientes**:
  - Identificador del paciente.
  - Iniciales del nombre y apellidos.
  - Sexo.
  - Fecha de nacimiento.
  - Semanas de gestación.
  - Peso al nacer.
  - Indicador de si existió riesgo prenatal.
  - Indicador de si existió riesgo perinatal.
  - Indicador de existencia de problemas de audición.
  - Indicador de existencia de antecedentes de patología neurológica.
  - Descripción de los antecedentes de patología neurológica (si los hubiera).
  - Indicador de existencia de antecedentes de patología general.
  - Descripción de los antecedentes de patología general (si los hubiera).
  - Estado del paciente en el sistema (en tratamiento o histórico).

- **Datos de las evaluaciones del lenguaje**:
    - Identificador de la evaluación.
    - Edad en meses del paciente en el momento de la evaluación.
    - Meses a los que corresponden las preguntas realizadas en la evaluación.
    - Lista de los hitos evaluados y de un indicador que revela si el hito se alcanzó o no.
    - Decisión o decisiones propuestas por el sistema para esta evaluación.
    - Decisión del profesional, en el caso de no aceptar la propuesta del sistema.
    - Fecha de la evaluación.
    - Tipo de resultado de la evaluación.
    - Duración en segundos de la evaluación en el sistema.
- **Datos de las valoraciones de las evaluaciones**:
    - Identificador de la evaluación.
    - Identificador del profesional que realizó la valoración.
    - Identificador del tipo de valoración.
    - Valoración respecto de si la decisión del sistema estaba indicada o no.
    - Fecha de validación.

Dado que lo se pretende es explotar los datos de las evaluaciones, los datos de la BD que utilizaremos para la generación de estadísticas son los siguientes:

- **Datos de los pacientes.**
- **Datos de las evaluaciones.**

Tenga en cuenta que, aunque no usaremos los datos de las valoraciones, estos también están relacionados con las evaluaciones y se propone su uso en futuras mejoras de los sistemas Pegaso y Gades.

Para permitir al usuario seleccionar un centro y un profesional al inicio del proceso de consulta de estadísticas, también se usarán los datos siguientes: **nombre y apellidos del profesional** que realizó la evaluación y el **centro** al que pertenece. Sin embargo, estos datos no se usarán en la elaboración de las estadísticas en sí.

Se definen las siguientes estadísticas:

1. **Incidencia de cada tipo de resultado por meses de las preguntas (en tanto por ciento)**. Esta gráfica mostrará el porcentaje de cada tipo de decisión dada por el

sistema (normal, aviso o alarma) por cada año (los años van del 1 al 6). Los años se calcularán a partir de los meses de las preguntas, considerando que las preguntas de los meses 1 al 12 forman parte del año 1, las de los meses 13 al 24 forman parte del año 2, y así sucesivamente.

2. **Incidencia de cada tipo de resultados por sexo**. Esta gráfica mostrará el número total de casos en los que ha ocurrido cada tipo de decisión del sistema (normal, aviso o alarma) por sexo (masculino, femenino y ambos conjuntamente).
3. **Incidencia de cada tipo de resultado por sexo (en tanto por ciento)**. Esta gráfica mostrará los mismos datos que la anterior, pero en tanto por ciento. Es decir, mostrará el porcentaje de ocurrencias de cada tipo de decisión del sistema (normal, aviso o alarma) por sexo (masculino, femenino y ambos conjuntamente).
4. **Bondad de la Base de Conocimiento por años (en tanto por ciento)**. Esta gráfica mostrara la bondad de la BC por cada año (años del 1 al 6). La bondad se define como el porcentaje de evaluaciones en las que los profesionales que las realizaron estaban de acuerdo con la decisión propuesta por el sistema. También se mostrará el porcentaje de evaluaciones en las que los profesionales no estaba de acuerdo con la decisión propuesta por el sistema. Los años se calcularán en base a los meses de las preguntas, tal y como se ha explicado en la estadística 1.
5. **Bondad de la Base de Conocimiento por etapa educativa (en tanto por ciento)**. Esta gráfica mostrará la bondad de la BC por etapa educativa de los niños. En este caso también se mostrará el porcentaje de evaluaciones en las que los profesionales que las realizaron no estaban de acuerdo con la decisión propuesta por el sistema. Los años se calcularán en base a los meses de las preguntas, tal y como se ha explicado en la estadística 1. Se distinguirán tres etapas educativas en la gráfica: de 1 a 3 años, de 4 a 6 años y ambas conjuntamente.
6. **Tabla resumen de las evaluaciones realizadas por el profesional seleccionado**. Esta opción no estará disponible cuando se consulten estadísticas de evaluaciones realizadas por más de un profesional. En esta tabla se mostrará un resumen de los datos más significativos de las evaluaciones realizadas por un profesional. Se mostrará, por este orden, la siguiente información de cada evaluación:
    - Fecha de la evaluación.
    - Sexo.
    - Fecha de nacimiento.
    - Meses del niño.
    - Meses de las preguntas.
    - Decisión propuesta por el sistema.

- ¿El pediatra aceptó la decisión propuesta por el sistema?
- Observaciones.

Elección del API para generar las gráficas

Como se puede deducir de lo que acabamos de explicar, trabajaremos con dos tipos de representaciones para las estadísticas:

- **Tablas**: se usará el lenguaje HTML para generarlas.
- **Gráficas**: se usará un API para generarlas.

Existen multitud de API que permiten generar gráficas en una aplicación web, pero antes de tomar una decisión debemos considerar los dos requisitos siguientes:

- Primero, uno de los requisitos funcionales definidos en el apartado 3.5.2 (el requisito RNF7) determina que nuestra elección ha de ser software libre y gratuito (tenga en cuenta que "software libre" no implica gratuidad).
- Segundo, el API elegido deberá basarse en un lenguaje que se estudie como parte de las materas obligatorias de los planes de estudios de nuestra Escuela, de modo que no sea necesario invertir tiempo en aprender un nuevo lenguaje.

Se han considerado varios API que cumplen la primera condición, es decir, que son software libre y gratuito:

- **JFreeChart**: este API permite generar una amplia gama de gráficas 2D (de barras, de líneas, de barras, circulares, etc.) en el lado del servidor. Algunas de estas gráficas están disponibles en 3D. Además, provee al usuario de herramientas para personalizar ampliamente el aspecto de las gráficas (colores, líneas, leyenda, fuentes de texto, etc.) y soporta diferentes formatos de salida para las gráficas (PNG, JPEG, PDF, etc.), así como componentes de la librería Java Swing. Está diseñado para ser usado en aplicaciones, *applets*, Servlets y JSP [JFR05]. La última versión se ha publicado en julio de 2014.
- **Cewolf** (*Chart Enabling Web Object Framework*). Este API está destinado a ser usado en aplicaciones web y se basa en JFreeChart, por lo que las gráficas y las posibilidades de personalización de las mismas son similares. Sin embargo, no está basado en Java, sino en etiquetas XML, y no se genera ningún archivo en el lado del cliente; todo se basa en objetos de sesión ligeros y análisis dinámico de datos [CEW02].
- **Google Charts**. Este API provee una forma de visualizar datos de forma interactiva en una página web, desde simples gráficas de línea hasta complejos árboles

jerárquicos. La manera más común de usarlo es embebiendo código JavaScript en la página web. Permite ejecutar sentencias SQL para acceder a una BD y las gráficas se representan usando HTML5/SVG (*Scalable Vector Graphics*) [GCH13].

- **RGraph**. Este API está basado en HTML5 y JavaScript. Permite diseñar gráficas interactivas para páginas web en el lado del cliente (navegador web). Usa la nueva etiqueta definida en el estándar HTML5: <canvas>. Esta etiqueta permite dibujar un mapa de bits que se controla mediante JavaScript (es decir, se dibuja en el elemento <canvas> usando JavaScript) [RGR14].

Nos hemos decantado por la primera opción, JFreeChart, ya que dicho API está basado en Java, que es un lenguaje que cumple nuestra segunda condición: se estudia en nuestra Escuela y su uso es muy extendido en bastantes asignaturas.

Por otra parte, JFreeChart es sencillo de utilizar y, dado que la velocidad no es un requisito crucial en nuestro caso, podemos generar las gráficas en el lado del servidor. Por último, otro factor a tener en cuenta es que existe una guía para el desarrollador [JFR07] muy completa que contiene numerosos ejemplos que serán de ayuda en la implementación de las diferentes gráficas.

3.4.4. Implementación

En este apartado vamos explicar qué componentes se han implementado en cada capa para hacer operacional la nueva funcionalidad.

Capa de presentación

En la capa de presentación se han añadido las vistas (páginas JSP) y los ayudantes de vista (Servlets) necesarios para llevar a cabo las tareas descritas por los casos de uso (ver figura 54).

Las nuevas **vistas** y sus responsabilidades asociadas son las mostradas en la tabla siguiente.

Vista	Caso de uso o responsabilidad
ConsultarEstadisticasForm	Selección de centro educativo y profesional
SeleccionarEstadisticasForm	Selección de la estadística que se desea consultar
MostrarEstadisticasForm	Muestra la gráfica o tabla correspondiente a la estadística seleccionada

Tabla 2. *Vistas de la funcionalidad de consulta de estadísticas*

El único caso de uso que no tiene una vista asociada es el de impresión de la estadística mostrada, ya que para ello se usará la sintaxis JavaScript *window.print()* en la vista *MostrarEstadisticasForm*.

Por otra parte, los nuevos **ayudantes de vista** (los ayudantes de vista soportan a las vistas realizando una serie de tareas adicionales) y sus responsabilidades son los siguientes:

Ayudante de vista	Responsabilidad
ConsultarEstadisticasHelper	Apoya a la vista *ConsultarEstadisticasForm* en la selección de centro educativo y profesional
SeleccionarEstadisticasHelper	Apoya a la vista *SeleccionarEstadisticasForm* en la selección de la estadística deseada
MostrarEstadisticasHelper	Apoya a la vista *MostrarEstadisticasForm* en la generación de la gráfica correspondiente a la estadística seleccionada

Tabla 3. *Ayudantes de vista de la funcionalidad de consulta de estadísticas*

La **sesión**, como explicamos en el apartado 2.2.2, es el contexto para el intercambio de datos entre vistas y ayudantes de vista. En la funcionalidad de consulta de estadísticas se usan los siguientes atributos de sesión:

Nombre de atributo	Tipo	Descripción
estadisticas	Collection<EstadisticaBean>	Conjunto de datos de las evaluaciones realizadas en el centro seleccionado y por los profesionales seleccionados
estadisticaSeleccionada	String	Estadística solicitada por el usuario
idCentro	Long	Identificador de centro del profesional cuyas estadísticas se están consultando
idPediatra	Integer	Identificador del profesional que realizó las evaluaciones cuyas estadísticas se están consultando
medicoEval	MedicoBean	Profesional que realizó las evaluaciones cuyas estadísticas se están consultando
nombreCentro	String	Nombre del centro al que pertenecen las estadísticas que se están consultando

Tabla 4. *Atributos de sesión usados en la funcionalidad de consulta de estadísticas*

El atributo de sesión "nombreCentro", como se ha explicado en el apartado 3.3.1, se ha añadido en el transcurso de este PFG. Del mismo modo, los atributos de sesión "estadisticas", "estadisticaSeleccionada" e "idPediatra" también han sido añadidos en este PFG, mientras que los atributos de sesión "idCentro" y "medicoEval" ya se definieron en [URI13].

Para poder entender mejor la interacción entre los componentes de la capa de lógica que acabamos de definir (vistas y ayudantes de vista), el **diagrama de secuencia** mostrado en la figura 56 puede resultar de ayuda. En él se muestra cómo interaccionan ambas clases de componentes durante la ejecución normal del proceso de consulta de estadísticas.

Observe que el ayudante de vista *MostrarEstadisticasHelper* solo interviene cuando se ha seleccionado una estadística que requiere la generación de una gráfica. Si, por el contrario, se necesita generar una tabla, ésta se creará en la vista *MostrarEstadisticasForm*. De ahí que se pueda leer "opcional" en dicha parte.

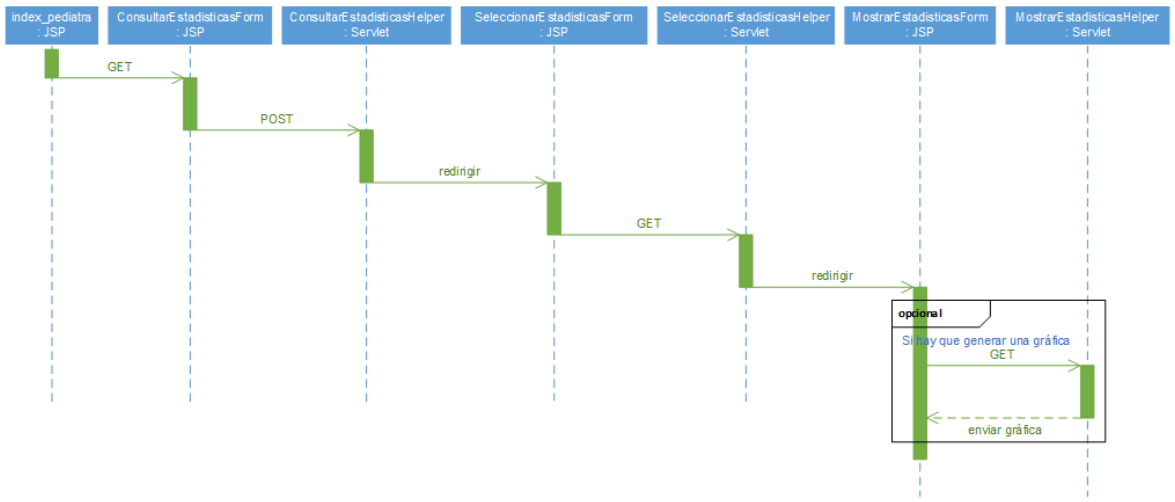

Figura 56. *Interacción en la capa de presentación de la consulta de estadísticas de Gades*

Capa de lógica

A pesar de que no se ha añadido ningún controlador de lógica adicional (ver apartado 2.2.2), sí se han modificado los controladores de lógica ya definidos (*EvaluacionLenguajeController* y *MedicosController*). En dichos controladores se han realizado los siguientes cambios:

- ***EvaluacionLenguajeController***: se ha añadido el método *public Collection<PacienteBean> obtenerPacientesPorMedico(MedicoBean medico)* para solicitar todos los pacientes (*Collection<PacienteBean>*) correspondientes al médico pasado como parámetro (*medico*).

- ***MedicosController***: se ha añadido el método *public CentroBean consultarCentroPorId(Long idCentro)* para solicitar un centro educativo (*CentroBean*) a partir del identificador pasado como parámetro (*idCentro*).

Por otra parte, también se ha añadido el componente JavaBean de transferencia de datos siguiente:

- ***EstadisticaBean***: este JavaBean se usa para agrupar en un mismo objeto los datos de un paciente, los datos de las evaluaciones (si hay varias) y las valoraciones de las decisiones del sistema de cada evaluación. Se usa para simplificar el tratamiento de datos de cara a presentarlos en las tablas o gráficas, según corresponda.
  Concretamente, este JavaBean se usa en el ayudante de vista *SeleccionarEstadisticasHelper* (Servlet) para almacenar los datos relativos a las evaluaciones de un paciente. En dicho ayudante de vista se crea una colección de tantos objetos *EstadisticaBean* como pacientes hayan sido evaluados por los profesionales seleccionados. Dicho objeto se almacena en la sesión (en el atributo de sesión "estadisticas") para la posterior elaboración de los datos estadísticos que se mostrarán en las tablas o gráficas que se generen.
  En la tabla 5 podemos ver los atributos de este JavaBean.

Atributo	Tipo	Descripción
paciente	PacienteBean	Paciente al que corresponden las evaluaciones contenidas en este objeto
evaluaciones	Collection<EvaluacionBean>	Evaluaciones del lenguaje del paciente
valoraciones	Collection<ValidacionBean>	Valoraciones de las decisiones del sistema correspondientes a las evaluaciones del lenguaje realizadas

Tabla 5. *Atributos del* JavaBean *de transferencia de datos* EstadisticaBean

  No hemos usado el atributo "valoraciones", pues, por las estadísticas que hemos definido, no lo hemos necesitado. Se decide dejar preparado este atributo por si se necesita en futuras mejoras.

Capa de datos

No se han añadido nuevas tablas a la BD. Sin embargo, sí se han añadido algunos métodos en los componentes JavaBean de acceso a datos (DAO):

- ***UtilidadesDAO***: en este JavaBean se ha añadido el método *public Collection<PacienteBean> findPacientesByMedico(Integer idMedico)*, que en el caso

de Gades permite obtener la colección de pacientes (*Collecion<PacienteBean>)* que han sido evaluados por el profesional cuyo identificador se pasa como parámetro (*idMedico*). Este método es invocado desde la capa de lógica por el nuevo método que hemos definido en el controlador de lógica *EvaluacionLenguajeController*.

- ***CentroMedicoDAO***: en este JavaBean se ha añadido el método *public CentroBean findCentroById(Long idCentro)* para obtener un centro educativo de la BD a partir de su identificador (*idCentro*). Este método será invocado desde la capa de lógica por el nuevo método que hemos definido en el controlador de lógica *MedicosController*.

Interacción entre capas

Para entender cómo interactúan unas capas con otras, podemos ver el diagrama mostrado en la figura 57, en el que aparece la interacción entre los componentes de todas las capas durante el proceso de consulta de estadísticas.

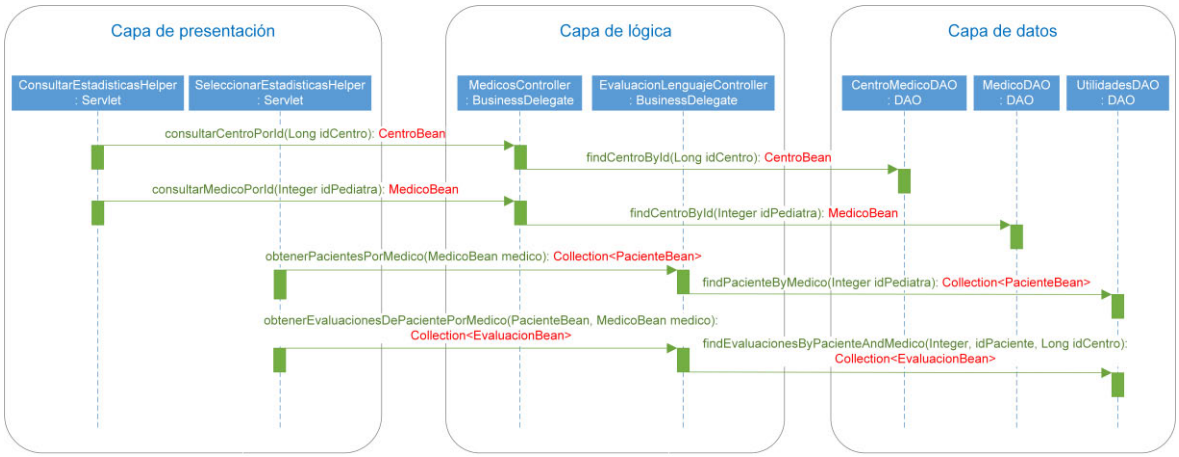

Figura 57. *Interacción entre los componentes de cada capa durante la consulta de estadísticas*

En este diagrama debemos observar lo siguiente:

- La interacción entre los ayudantes de vista, que pertenecen a la capa de presentación, y los delegados de negocio, o controladores de lógica, que pertenecen a la capa de lógica. Dicho de otro modo, los ayudantes de vista usan los métodos ofrecidos de los controladores de lógica.
- Los componentes de la capa de lógica hacen a su vez uso de los componentes de la capa de datos: objetos de acceso a datos (DAO). Es decir, los controladores de lógica usan los métodos ofrecidos por los DAO. Aunque no se incluye en la figura, hay que tener en cuenta que los DAO obtienen los datos de la BD usando sentencias SQL.
- Los objetos devueltos por los métodos, es decir, los objetos que se envían de la capa de datos a la capa de lógica y de ésta a la capa de presentación son objetos JavaBean

de transferencia de datos o colecciones de los mismos. Estos objetos se han marcado en color rojo en la figura 57.

Gestión de errores

Cuando en la aplicación se produce un error, se lanza una **excepción**. Éste es el mecanismo empleado por Java para gestionar errores y otras situaciones excepcionales. Esta excepción debe ser capturada por el método que detectó la excepción, quien tiene dos posibilidades:

- **Tratar la excepción**, es decir, dejar un registro en los archivos de bitácora (*log*) de la aplicación.
- **Lanzar una nueva excepción** para que sea capturada por el objeto que invocó el método en cuestión, es decir, dejar que sea un componente de una capa superior quien decida qué hacer con dicha excepción. Esto es lo que se conoce como propagar una excepción.

En nuestro caso el tratamiento concreto que se da a las excepciones que se producen en la funcionalidad de consulta de estadísticas es el mismo que se definió en [URI13] para el resto de funcionalidades de la aplicación. También se emplean las mismas excepciones definidas en dicho trabajo.

Como ejemplo, supongamos que realizamos un proceso de consulta de estadísticas y que, tras seleccionar una estadística, no se muestra una gráfica o tabla en pantalla, sino un mensaje de error. Suponemos también que esta excepción se ha producido al ejecutar una sentencia SQL para obtener un centro de la BD. La secuencia de acciones que se ha producido, y que aparece explicada gráficamente en la figura 58, es la siguiente:

1. En la capa de datos, el método *public MedicoBean findPacientesByMedico(Integer idMedico)* de la clase *UtilidadesDAO* captura una excepción *SQLException* y lanza una excepción *DAOException.*
2. En la capa de lógica, la excepción *DAOException* es capturada por el método: *public Collection<PacienteBean> obtenerPacientesPorMedico(Integer idMedico)* de la clase *EvaluacionLenguajeController.* Este método lanza una excepción de tipo *ConsultaMedicoException.*
3. En la capa de presentación, la excepción *ConsultaMedicoException* es capturada por el método *protected void processRequest(HttpServletRequest request, HttpServletResponse response)* de la vista de usuario *SeleccionarEstadisticasHelper.* Este método lanza una excepción de tipo *ErrorValidacionException.*

4. La excepción *ErrorValidacionException* está asociada a la vista *ErrorValidacionView* (JSP), por lo que se muestra un mensaje de error por pantalla que contendrá una descripción del error producido.

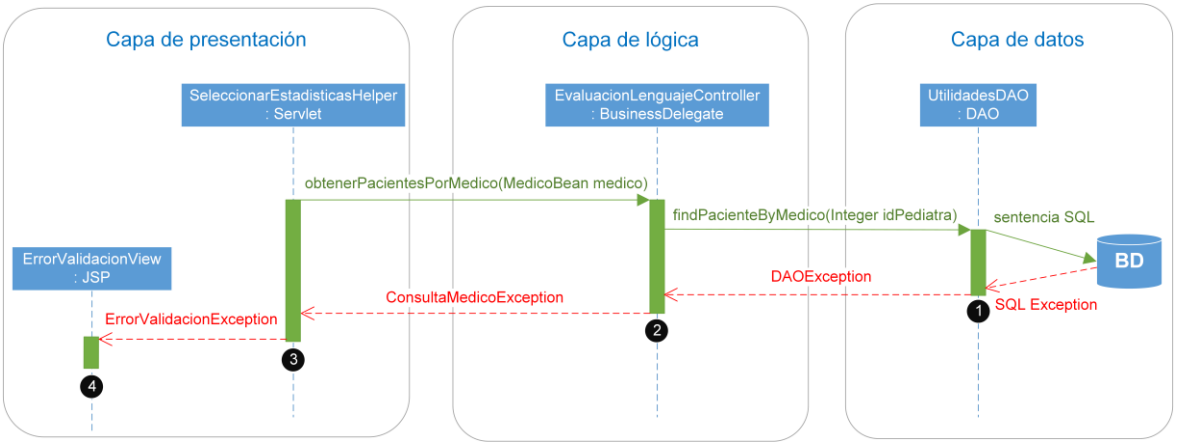

Figura 58. *Gestión de errores en Gades*

4. Pruebas y resultados

Una vez descritas las tareas llevadas a cabo en este proyecto, en este capítulo vamos a exponer las pruebas realizadas y a analizar los resultados obtenidos en las mismas.

Se han dividido las pruebas en varios apartados, de acuerdo con las tareas detalladas en el capítulo 3:

- Funcionamiento de Pegaso y Gades con Java SE 7 (apartado 4.1).
- Aplicación para generar ontologías OWL desde código Java (apartado 4.2).
- Mejoras realizadas sobre las funcionalidades existentes en Pegaso y Gades:
    - Opción para imprimir un informe de consulta de resultados (apartado 4.3.1).
    - Dependencia con el URI de la ontología en la clase *EvaluacionLenguajeController* (apartado 4.3.2).
    - Cambio en la lógica de negocio que transforma las respuestas "NS/NC" en "NO" (apartado 4.3.3).
    - Adición del símbolo de *copyright* de la UPM (apartado 4.3.4).
- Funcionalidad para la consulta de estadísticas (apartado 4.4).

Las pruebas realizadas servirán para comprobar el correcto funcionamiento de los sistemas Pegaso y Gades. Estas pruebas se realizan para un solo sistema en cada apartado, pero son extrapolables de uno a otro sistema, ya que Pegaso y Gades tienen la misma arquitectura y son prácticamente idénticos desde el punto de vista funcional.

4.1. Funcionamiento de Pegaso y Gades con Java SE 7

En este apartado vamos a comprobar si el problema de la incompatibilidad con Java SE 7, que se ha explicado en el apartado 3.1, ha sido resuelto tras añadir el fichero *setenv.bat* en el directorio *~/apache-tomcat-7.0.50/bin/* del servidor Apache Tomcat (ver apartado 3.1.4).

Los pasos que hemos seguido para realizar las pruebas que permitan llevar a cabo esta comprobación son los siguientes:

1. En primer lugar, hemos consultado la BC en busca de meses cuyos hitos alberguen al menos un hito de la clase "HitoAlarma", que son los que ocasionaban el problema (los hitos de la clase "HitoAviso" no ocasionan problema alguno, ya que no

requieren la realización de un proceso de inferencia). Hemos seleccionado varios de estos meses, a los que denominaremos "meses de interés".

2. A continuación, hemos realizado varias evaluaciones para niños de ejemplo cuyas fechas de nacimiento sean tales que las preguntas realizadas por el proceso de evaluación del lenguaje se correspondan con los hitos de los meses de interés. Para ello, hemos introducido una fecha de nacimiento para cada niño que sea igual a la diferencia entre la fecha en la que se realiza cada evaluación y el valor numérico de cada mes de interés. Además, hemos elegido siempre una semana de gestación superior a la 36. Y hemos respondido "no" a todas las preguntas, para forzar la realización de un proceso de inferencia.
3. Por último, hemos observado que las evaluaciones del lenguaje se realizan con normalidad, es decir, aparece en cada una de ellas una decisión del sistema por pantalla en lugar del mensaje de error que aparecía anteriormente (ver figura 22).

Como ejemplo, exponemos a continuación una de las pruebas realizadas:

1. Escogemos como mes de interés el mes 1, cuyos dos hitos son de la clase "HitoAlarma" (ver figura 43). Por tanto, será necesario llevar a cabo un proceso de inferencia.
2. Realizamos la evaluación del lenguaje de una niña de ejemplo cuya fecha de nacimiento es tal que las preguntas realizadas por el sistema corresponden a los dos hitos del mes 1. Para ello, restamos a la fecha de la evaluación, 24/07/2014, el número del mes de interés (mes 1), quedando como fecha de nacimiento 24/06/2014. Además, elegimos 37 semanas en el campo "semanas de gestación". Podemos ver los datos de esta niña de ejemplo en la figura 59.

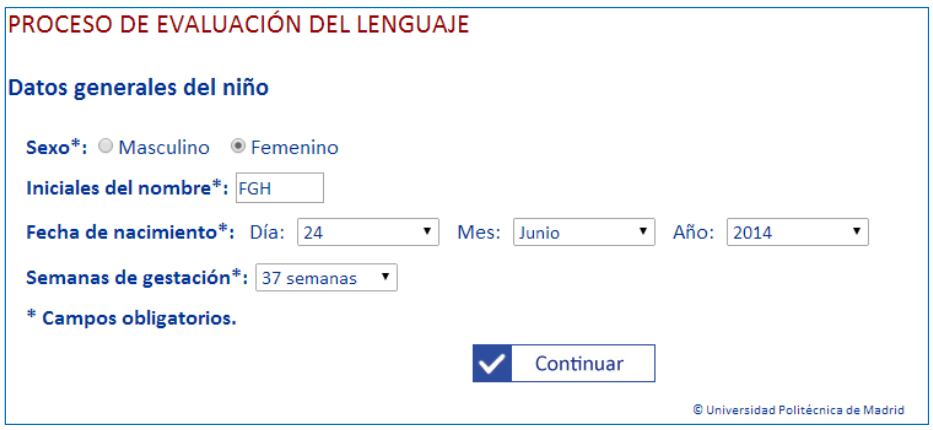

Figura 59. Datos de la niña de ejemplo para la realización de una inferencia

3. Comprobamos que tras responder "no" a las dos preguntas asociadas a los hitos del mes 1 (ver figura 60), se muestra un resultado por pantalla (en lugar del mensaje de

error que se mostraba anteriormente). El resultado de la evaluación se puede ver en la figura 61: "Enviar al especialista para comprobar audición".

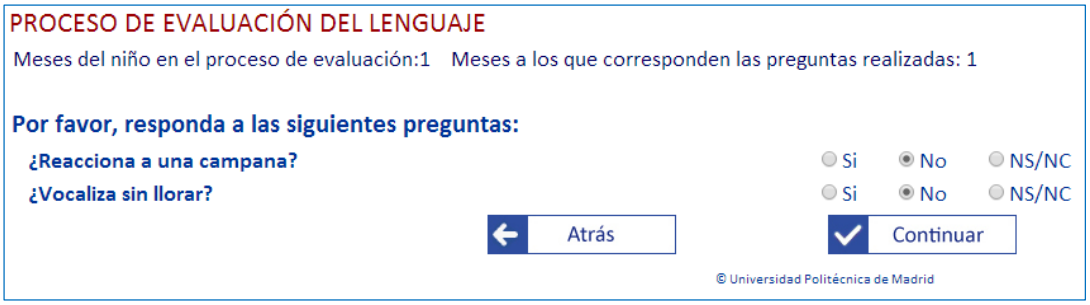

Figura 60. Preguntas de la evaluación de la niña de ejemplo para realizar una inferencia

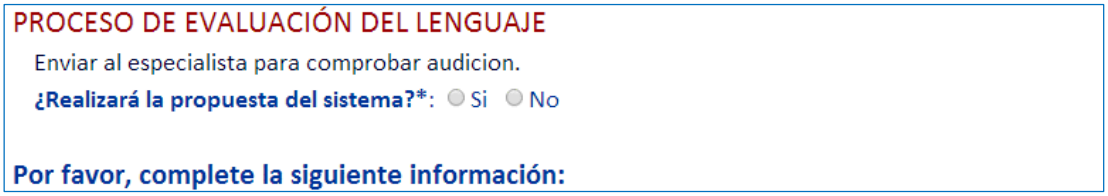

Figura 61. Resultados de la evaluación de la niña de ejemplo tras la inferencia realizada

Hemos guardado esta evaluación para consultarla más adelante en otras pruebas.

En definitiva, podemos considerar que el problema de incompatibilidad de Pegaso y Gades con Java SE 7 ha sido resuelto y, por tanto, la solución propuesta es válida.

4.2. Aplicación para generar ontologías OWL desde código Java

En este apartado comprobaremos si la ontología OWL generada usando la aplicación descrita en el apartado 3.2 es válida. Es decir, comprobaremos si es posible realizar una inferencia con éxito para un proceso de evaluación del lenguaje usando dicha ontología, que se puede encontrar en el Anexo I.

En primer lugar, hemos añadido la ontología reducida (*ontologiaMes1.owl*) en el directorio *~\apache-tomcat-7.0.50\webapps\pegaso\owl* del servidor Apache Tomcat, que es donde se sitúa el archivo de la ontología OWL. Puede haber varias ontologías en dicho directorio, pero sólo se usará una. La ontología usada será la que se indique en el archivo de propiedades *Configuration.properties*, que define los nombres y valores de ciertas propiedades que son leídas y utilizadas por la aplicación [URI13]. Una de estas propiedades es "OWLFile", que es la ruta del archivo OWL que contiene la ontología.

En la figura 62 se puede ver, recuadrado en color rojo, el valor dado a dicha propiedad para hacer las pruebas.

```
# DataBase UserdataBase
DBUser=oscar
# DataBase Password
DBPass=oscar
# DataBase connection
Conn=jdbc:mysql://localhost:3306/pegasodb
# DataBase Driver
Driver=com.mysql.jdbc.Driver
# File OWL
OWLFile=/owl/ontologiaMes1.owl
```

Figura 62. *Fragmento del archivo* Configuration.properties *de Pegaso*

Puesto que la ontlogía OWL generada solamente cubre los hitos y la decisión del sistema que corresponden al mes 1 (año 1), hemos realizado una evaluación del lenguaje en Pegaso usando la misma niña de ejemplo que se ha usado en el apartado 4.2 (ver figura 59), que tiene 1 mes de edad en el momento actual. Así, hemos respondido a las mismas preguntas (ver figura 60), ya que la fecha de la realización de esta prueba ha sido la misma que la del apartado 4.2.

Los resultados obtenidos han sido los mismos resultados que los obtenidos en el apartado 4.2 (ver figura 61). Por tanto, podemos afirmar que la ontología generada por la aplicación es válida y permite realizar inferencias para el mes 1.

Además, hemos probado que la ontología sólo sirve para el mes 1 realizando inferencias para otros meses, en cuyo caso no hemos obtenido ninguna decisión por parte del sistema.

4.3. Mejoras sobre las funcionalidades existentes en Pegaso y Gades

En este apartado se explican las pruebas realizadas para verificar el correcto funcionamiento de las mejoras realizadas sobre las funcionalidades existente. Se ilustran solamente las pruebas realizadas en Gades, ya que las pruebas realizadas en Pegaso son iguales.

4.3.1. Opción para imprimir un informe de consulta de resultados

Para comprobar que la opción de imprimir un informe en la funcionalidad de consulta de resultados funciona correctamente, realizaremos un proceso de consulta de resultados para ver los resultados de la evaluación del lenguaje realizada para la niña de ejemplo en el apartado 4.1. El resultado de la consulta se muestra en la figura 63.

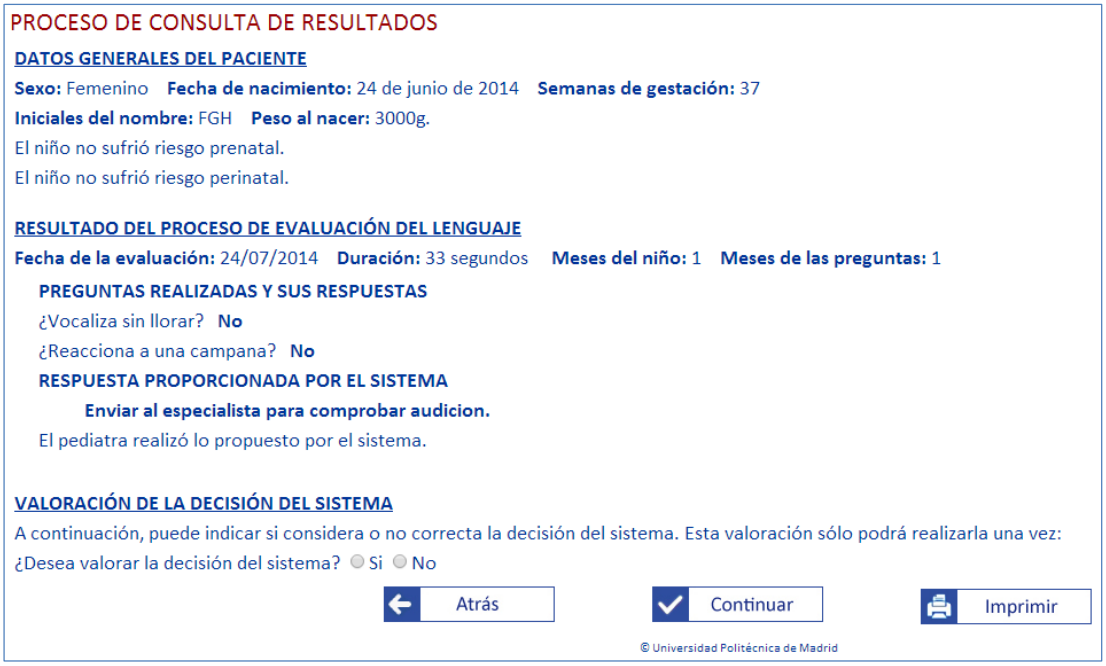

Figura 63. *Información adicional del proceso de evaluación del lenguaje de la niña de ejemplo*

Esta información es la que se debe mostrar en el informe, tras pulsar el botón "Imprimir" que aparece en la parte inferior derecha de la pantalla mostrada en la figura 63.

De acuerdo a lo explicado en el apartado 3.3.1, el formulario se muestra en una ventana emergente nueva que nos permite visualizarlo y enviarlo a la impresora, como podemos ver en la figura 64. Comparando los datos mostrados en la figura 64 con los mostrados en la figura 63, vemos que no falta ningún dato, por lo que podemos afirmar que el informe se ha generado correctamente.

Figura 64. *Vista de impresión de resultados en la funcionalidad de consulta de resultados*

Por último, si pulsamos el botón "Imprimir" que se muestra en parte inferior de la ventana mostrada en la figura 64, nos aparece el cuadro de diálogo para enviar el formulario a la impresora. En este cuadro de diálogo, que podemos ver en la figura 65, se puede seleccionar la impresora y las opciones de impresión (número de copias, color, márgenes, etc.).

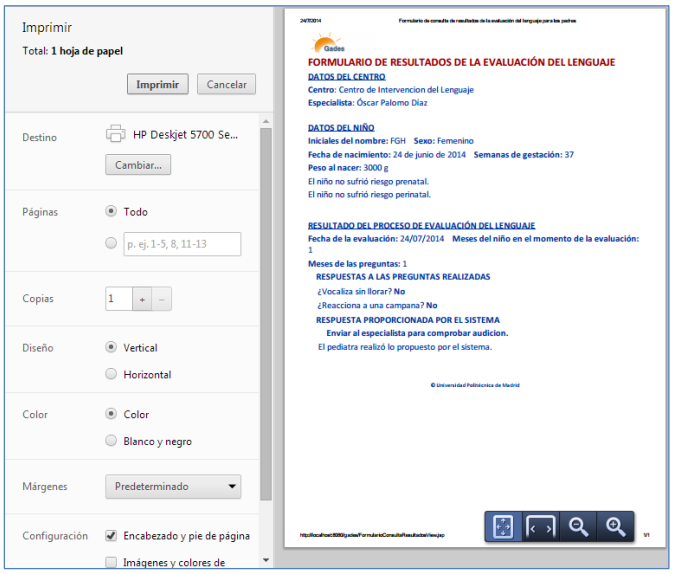

Figura 65. *Cuadro de diálogo de impresión del informe de consulta de resultados*

Una vez elegida la impresora (*HP Deskjet 5700*, en nuestro caso, como se puede ver en la figura 65) y las opciones de impresión, hemos conseguido el formulario impreso en papel. Hay que tener en cuenta que el cuadro de diálogo mostrado en la figura 65 corresponde al navegador web Google Chrome; dicho cuadro es ligeramente diferente en otros navegadores.

A continuación, hemos intentado guardar el formulario como PDF en el disco duro del ordenador, en lugar de imprimirlo, como se puede ver en la figura 66.

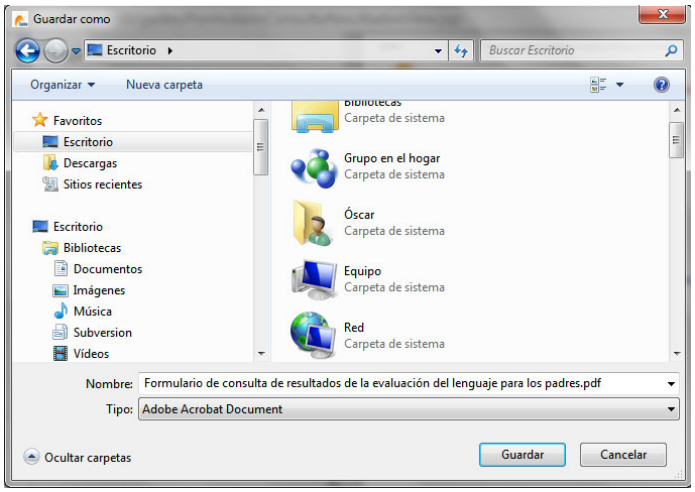

Figura 66. *Cuadro de diálogo mostrado al guardar el formulario como PDF*

La operación se ha realizado con éxito. Nuevamente, hay que tener en cuenta que este cuadro de diálogo corresponde al sistema operativo Windows 7, por lo que puede ser diferente en otros sistemas operativos.

Por tanto, podemos considerar que la opción para imprimir un informe de resultados en la funcionalidad de consulta de resultados funciona correctamente.

4.3.2. Dependencia con el URI de la ontología en la clase *EvaluacionLenguajeController*

En este apartado vamos a comprobar si se ha eliminado la dependencia con el URI de la ontología OWL, problema que se ha explicado en el apartado 3.3.2.

Para ello, primero hemos cambiado el valor del atributo *xmlns* ubicado en la cabecera de la ontología OWL usada en Pegaso. A continuación hemos modificado el valor de la propiedad "OWLFile" del archivo *Configuration.properties*, como explicamos en el apartado 4.2 (ver figura 62), para que la aplicación pueda encontrar el fichero de extensión .owl. Por último, hemos realizado varias evaluaciones del lenguaje para comprobar que el sistema infiere correctamente con diferentes URI para el espacio de nombres de la ontología.

Tras realizar varias veces el procedimiento explicado en el párrafo anterior, podemos afirmar que ya se ha resuelto la dependencia explicada en el apartado 3.3.2. Sea cual sea el URI del espacio de nombres de la ontología, el sistema infiere con normalidad. En la tabla 6 se muestran varios de los URI que se han probado.

Valores dados al atributo *xmlns*
http://www.owl-ontologies.com/Ontology1302172874.owl
http://ontologiaMes1.owl
http://www.owl-ontologies.com/pegaso.owl

Tabla 6. *Valores de* xmlns *probados para comprobar la eliminación de la dependencia con el URI*

4.3.3. Cambio en la lógica de negocio que transforma las respuestas "NS/NC" en "NO"

En este apartado vamos a comprobar si el problema explicado en el apartado 3.3.3 ha sido resuelto. Recordamos que el problema consistía en que las respuestas negativas ("no") en las preguntas relacionadas con los hitos de la clase "HitoAviso" no se almacenaban como "no" en la BD, sino como "ns/nc".

En primer lugar, hemos consultado la BC de Pegaso para encontrar un mes cuyos hitos sean subclases de la clase "HitoAviso" y hemos escogido el mes 2 (año 1). Dicho mes consta de los dos hitos que se pueden ver en la figura 67, ambos de tipo aviso.

Hito	Descripción (pregunta que el usuario del sistema responderá con objeto de evaluar el estado de adquisición del lenguaje en el niño)	Decisión Sistema Neuropediatra
2 meses - Aviso	Emite "OOO/AAH"	Adelantar visita (en tres meses)
2 meses - Aviso	Chilla y llora para interaccionar	Adelantar visita (en tres meses)

Figura 67. *Fragmento de la BC con los hitos de desarrollo correspondientes al mes 2 [MAR11]*

Tras realizar una evaluación de prueba con un niño de ejemplo de 2 meses de edad (en el momento de realizar la prueba) y responder "no" a las dos preguntas asociadas a los dos hitos anteriores, hemos accedido a la BD desde línea de comandos para comprobar que las respuestas almacenadas han sido las mismas que hemos introducido al realizar la evaluación.

Una vez dentro de la BD de Pegaso, hemos consultado la tabla "evaluacion". El fragmento que contiene la información correspondiente a la evaluación realizada es el que aparece en la figura 68. En rojo se han recuadrado las respuestas a las preguntas de la evaluación que se ha realizado (campo "detalle_estado_lenguaje"). Vemos que estas respuestas son "no", que son las respuestas que hemos dado al realizar la evaluación.

```
mysql> select * from evaluacion WHERE id_evaluacion='15';
+---------------+------------+-----------------+--------------------------+-----------------------+
| id_evaluacion | meses_niño | meses_preguntas | detalle_estado_lenguaje  | decision_sistema      |
+---------------+------------+-----------------+--------------------------+-----------------------+
|            15 |          2 |               2 | 7:No;8:No;               | ProximaVisitaEn3Meses; |
+---------------+------------+-----------------+--------------------------+-----------------------+

+------------------+-------------------+------------------+---------------+--------------------+
| acepta_propuesta | decision_pediatra | fecha_evaluacion | tipo_resultado | segundos_empleados |
+------------------+-------------------+------------------+---------------+--------------------+
| Si               |                   | 24/07/2014       | Aviso         |                 56 |
+------------------+-------------------+------------------+---------------+--------------------+
```

Figura 68. *Consulta realizada a la tabla "evaluacion" de la BD*

Por tanto, podemos afirmar que este problema ha sido resuelto con éxito.

4.3.4. Adición del símbolo de *copyright* de la UPM

En este apartado se comprueba que el símbolo de *copyright*, seguido del nombre de la universidad, se muestra correctamente en todas las vistas, como se explicó en el apartado 3.3.4.

Para ello, no hemos tenido más que recorrer todas las páginas JSP de la aplicación usando el navegador web, y comprobando que en la parte inferior de las mismas se muestra el texto deseado. Como muestra, se puede ver en la figura 69 una de las vistas con el símbolo de *copyright* y el texto añadido.

Figura 69. Vista inicial de Gades con el símbolo de copyright *añadido*

4.4. Funcionalidad para la consulta de estadísticas

En este apartado vamos a comprobar que la funcionalidad de consulta de estadísticas desarrollada, que se explicó en el apartado 3.4, funciona de acuerdo a lo esperado.

En primer lugar, realizaremos varios procesos de consulta de estadísticas para comprobar que las estadísticas definidas se muestran correctamente (apartado 4.4.1). Tras ello, evaluaremos si la nueva funcionalidad cumple los requisitos funcionales y no funcionales definidos durante la fase de diseño de la misma (apartado 4.42).

4.4.1. Realización de procesos de consulta de estadísticas

La primera vista que aparece tras acceder a la funcionalidad de consulta de estadísticas es *ConsultarEstadisticasForm*. En esta vista, como se decidió en el apartado 3.4.3, se puede seleccionar un profesional concreto de un centro médico (ver figura 70); todos los profesionales de un centro, escogiendo la opción "Todos los profesionales" en la lista desplegable de profesionales (ver figura 71); y también se pueden seleccionar todos los profesionales de todos los centros, escogiendo para ello la opción "Todos los centros" en la lista desplegable de centros (ver figura 72).

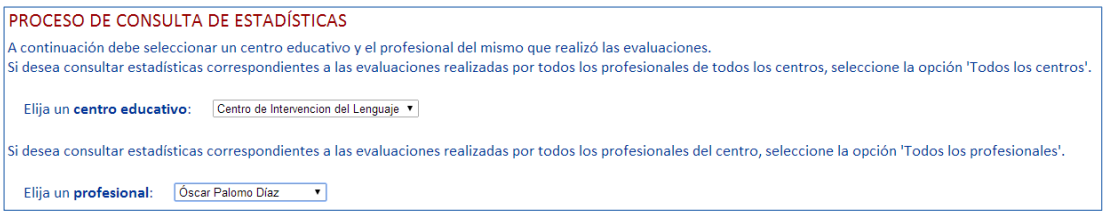

Figura 70. Vista ConsultarEstadisticasForm *del proceso de consulta de estadísticas (1/3)*

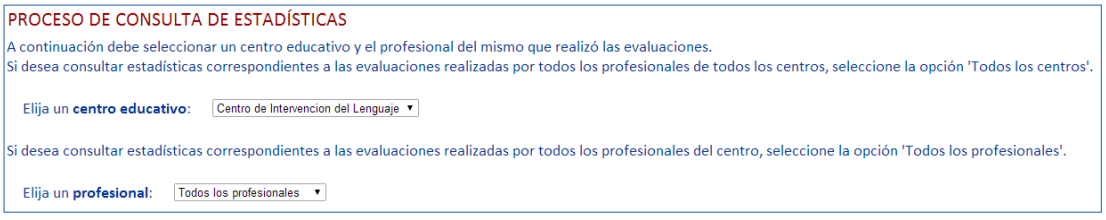

Figura 71. Vista ConsultarEstadisticasForm *del proceso de consulta de estadísticas (2/3)*

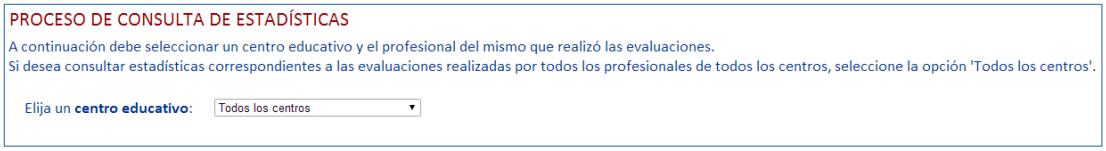

Figura 72. *Vista* ConsultarEstadisticasForm *del proceso de consulta de estadísticas (3/3)*

A continuación, se muestra la vista *SeleccionarEstadisticasForm*, donde el usuario puede seleccionar una estadística de entre las seis definidas en el apartado 3.4.3.

Las opciones que se muestran si la opción seleccionada en la vista anterior ha sido un único profesional son las mostradas en la figura 73.

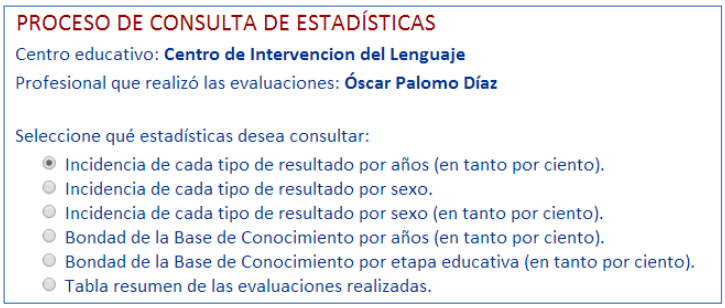

Figura 73. *Vista SeleccionarEstadisticasForm del proceso de consulta de estadísticas (1/3)*

Si, por el contrario, en la vista anterior se seleccionaron todos los pediatras de un centro o todos los pediatras de todos los centros, las opciones mostradas no incluyen la "Tabla resumen de las evaluaciones realizadas por el profesional seleccionado", como podemos ver en las figura 74 y 75.

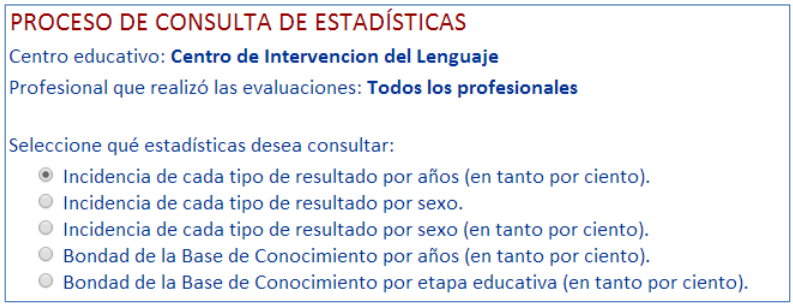

Figura 74. *Vista SeleccionarEstadisticasForm del proceso de consulta de estadísticas (2/3)*

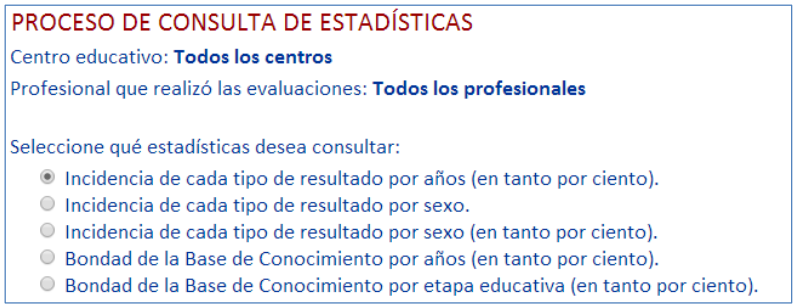

Figura 75. *Vista SeleccionarEstadisticasForm del proceso de consulta de estadísticas (3/3)*

Una vez seleccionada la estadística que se desea consultar, se mostrará en pantalla en la vista *MostrarEstadisticasForm*. En las figuras sucesivas, de la 76 a la 81, podemos ver el aspecto final de cada una de las opciones mostradas en la vista *SeleccionarEstadisticasForm* (ver figura 73). Tenga en cuenta que los datos mostrados en ellas no son reales; han sido elegidos al azar para la realización de las pruebas.

Incidencia de cada tipo de resultado por años (en tanto por ciento)

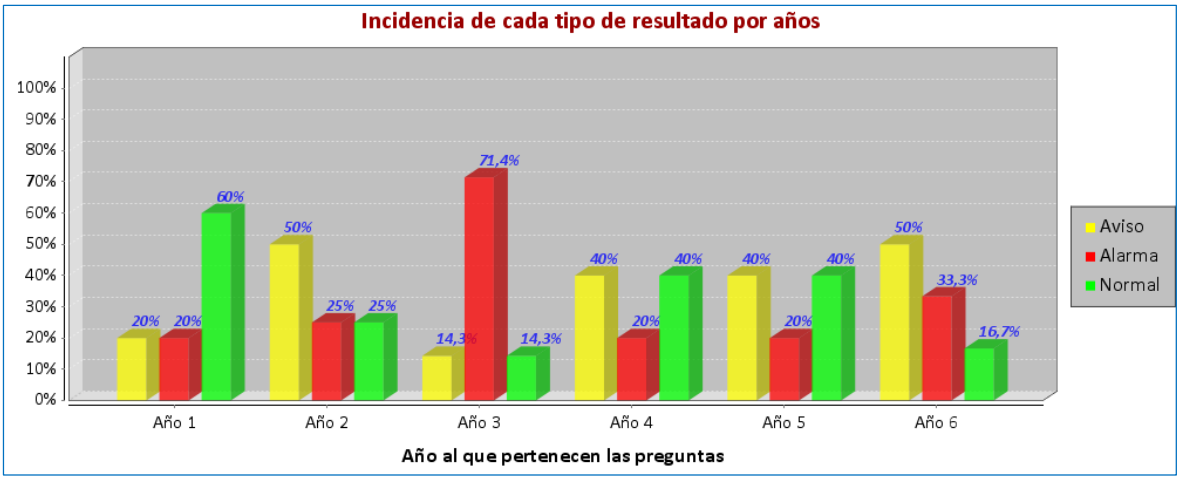

Figura 76. *Estadística mostrada en la vista MostrarEstadisticasForm (1/6)*

Incidencia de cada tipo de resultado por sexo

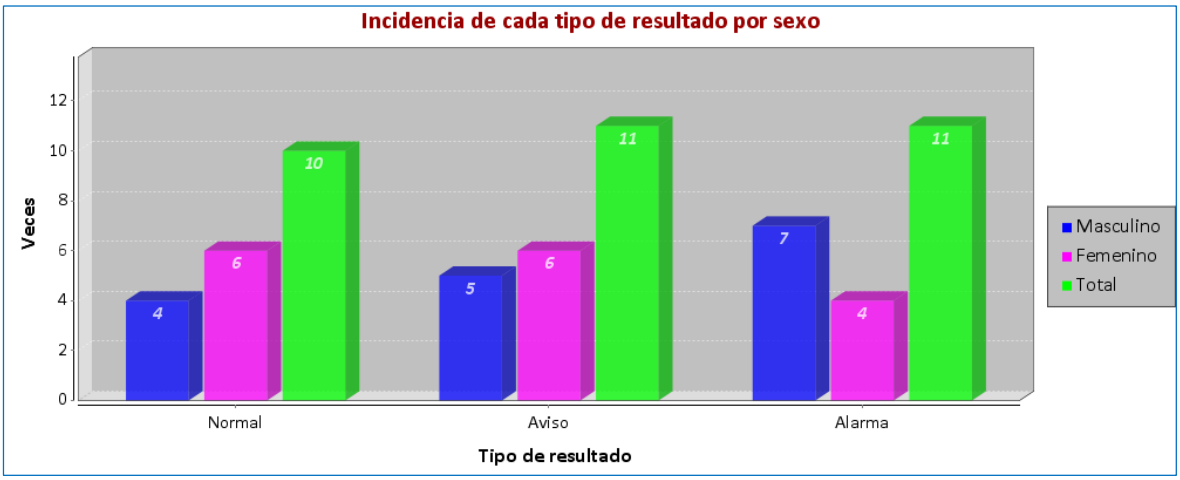

Figura 77. *Estadística mostrada en la vista MostrarEstadisticasForm (2/6)*

Incidencia de cada tipo de resultado por sexo (en tanto por ciento)

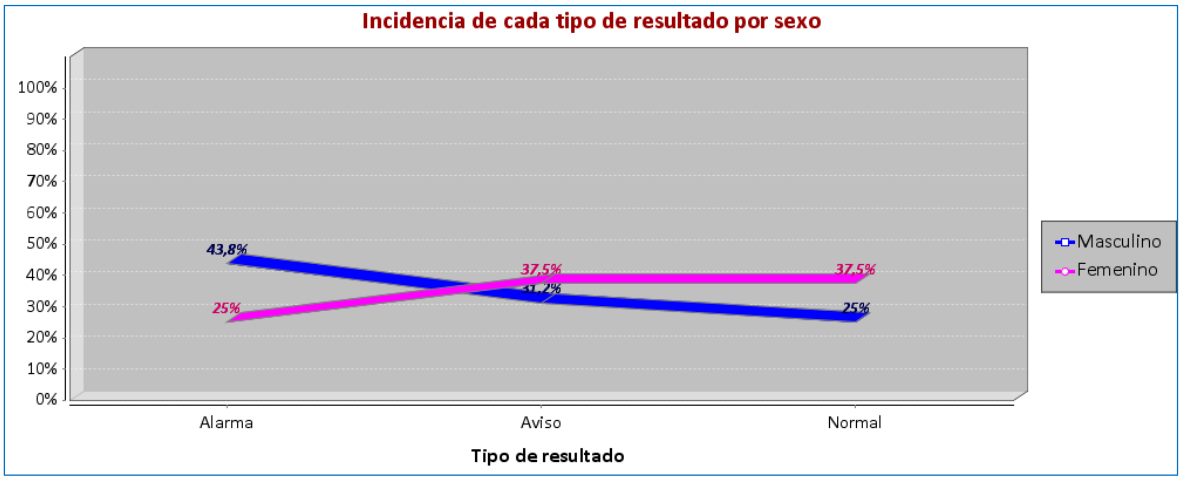

Figura 78. Estadística mostrada en la vista MostrarEstadisticasForm (3/6)

Bondad de la Base de Conocimiento por años (en tanto por ciento)

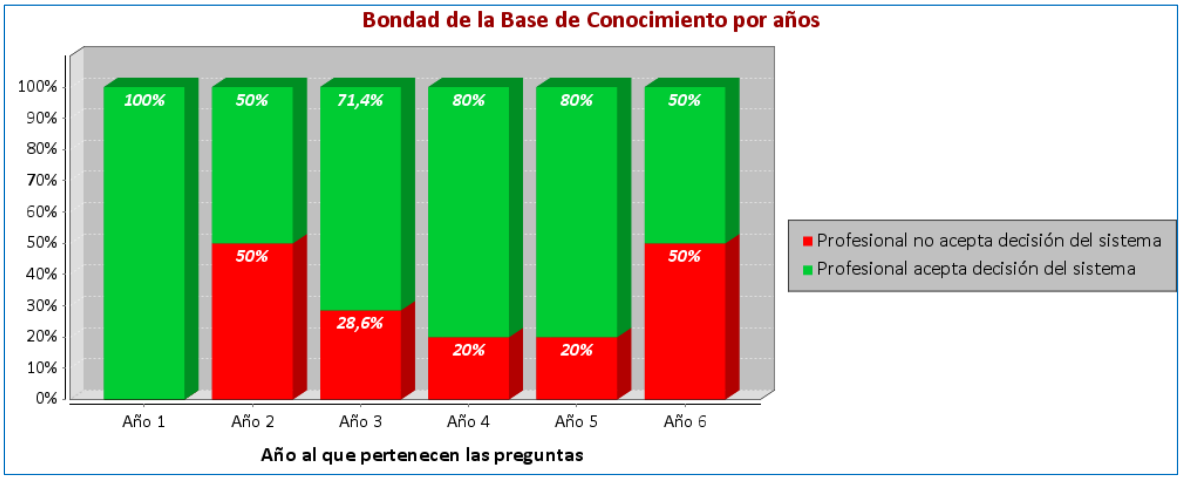

Figura 79. Estadística mostrada en la vista MostrarEstadisticasForm (4/6)

Bondad de la Base de Conocimiento por etapa educativa (en tanto por ciento)

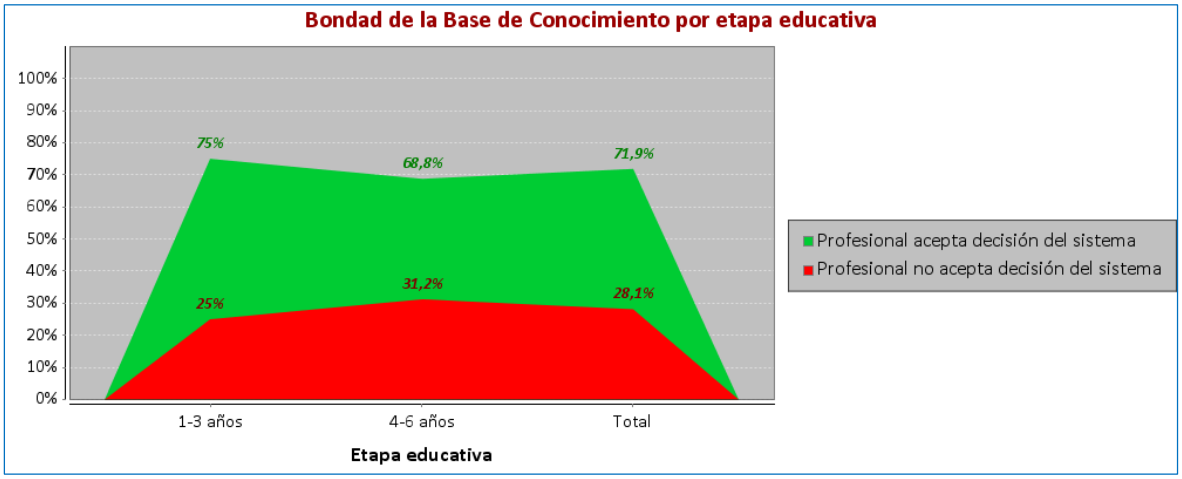

Figura 80. Estadística mostrada en la vista MostrarEstadisticasForm (5/6)

Tabla resumen de las evaluaciones realizadas

Resumen de las evaluaciones realizadas

Fecha de evaluación	Sexo	Fecha de nacimiento	Meses niño	Meses preguntas	Decisión del sistema	Acepta decisión	Observaciones
25/07/2014	Masculino	19 de septiembre de 2010	46	46	Próxima visita en 1 mes.	No	Objetable
25/07/2014	Masculino	20 de septiembre de 2009	58	54	Resultado normal.	Sí	
25/07/2014	Femenino	15 de septiembre de 2008	70	69	Próxima visita en 1 mes.	No	No estoy de acuerdo
25/07/2014	Femenino	1 de septiembre de 2013	9	9	Resultado normal.	Sí	
25/07/2014	Femenino	5 de septiembre de 2013	10	10	Resultado normal.	Sí	
25/07/2014	Femenino	22 de diciembre de 2011	31	30	Derivar a neuropediatra. Derivar a atención temprana.	Sí	
25/07/2014	Femenino	18 de septiembre de 2010	46	46	Próxima visita en 1 mes.	Sí	
25/07/2014	Femenino	20 de septiembre de 2010	46	46	Resultado normal.	Sí	
25/07/2014	Femenino	21 de septiembre de 2009	58	54	Próxima visita en 1 mes.	Sí	
25/07/2014	Femenino	20 de septiembre de 2008	70	69	Resultado normal.	Sí	

Figura 81. *Estadística mostrada en la vista MostrarEstadisticasForm (6/6)*

Por último, se pulsando el botón "Imprimir" en la vista *MostrarEstadisticasForm* aparece el cuadro de diálogo mostrado en la figura 82, que permite imprimir la gráfica o tabla mostrada en pantalla.

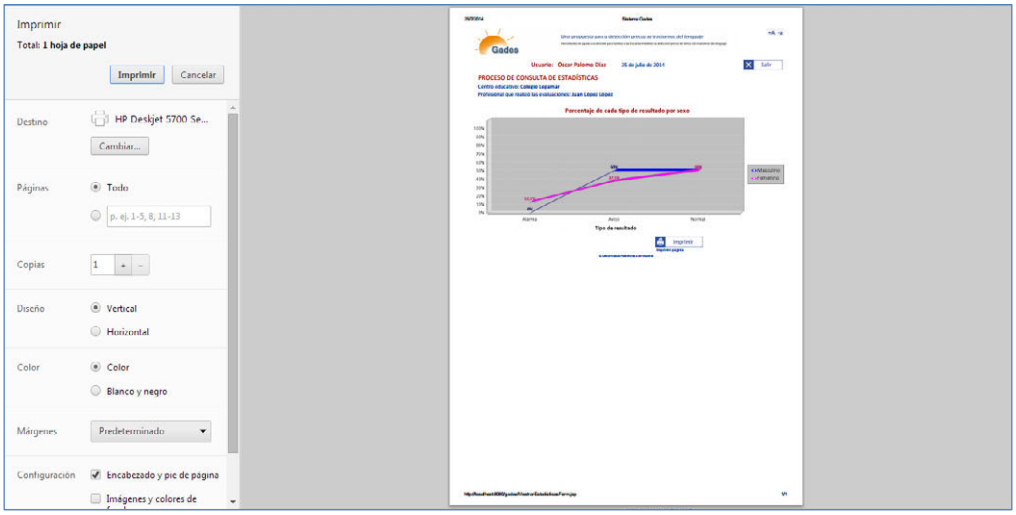

Figura 82. *Impresión de la estadística mostrada en la vista MostrarEstadisticasForm*

En cuanto a la **gestión de errores**, si el usuario de la aplicación no ha seleccionado un centro educativo ni un profesional en la vista inicial del proceso de consulta de estadísticas (*ConsultarEstadisticasForm*) se mostrará en pantalla el mensaje de error que aparece en la figura 83.

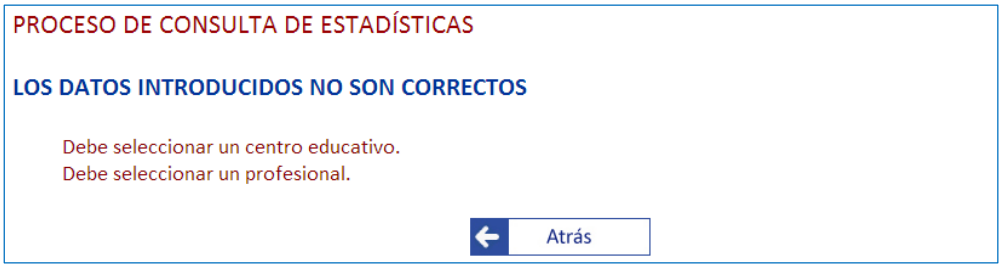

Figura 83. *Mensaje de error mostrado en caso de no seleccionar centro ni profesional*

Por otro lado, en caso de que los profesionales seleccionados aún no hayan realizado ninguna evaluación del lenguaje, se mostrará en pantalla el mensaje de error que aparece en la figura 84.

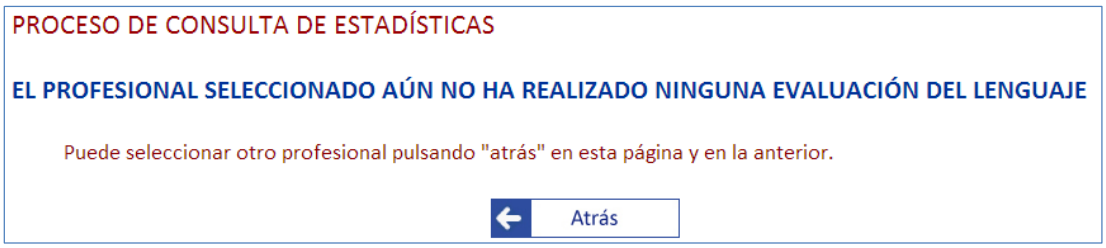

Figura 84. *Mensaje de error mostrado si el profesional no ha realizado ninguna evaluación*

4.4.2. Verificación de los requisitos funcionales y no funcionales

En la tabla siguiente (tabla 7) verificamos el cumplimiento de los requisitos funcionales definidos en el apartado 3.4.2.

Requisito	¿Se cumple?	Observaciones
RF1	Sí	Se ha realizado un proceso de consulta de estadísticas de manera exitosa empleando tres usuarios diferentes: un educador, un logopeda y el administrador.
RF2	Sí	Ver figura 70.
RF3	Sí	Ver figuras 70 y 71.
RF4	Sí	Ver figura 73.
RF5	Sí	Ver figuras 76, 77, 78, 79, 80 y 81.
RF6	Sí	Ver figura 82.
RF7	Sí	Ver figura 83.
RF8	Sí	Ver figura 84.
RF9	Sí	Ver figura 81.
RF10	Sí	Ver figuras 76, 77, 78, 79 y 80. Se considera que sí se cumple este requisito, aunque algunos colores podrían ser considerados chillones.

Tabla 7. *Verificación de los requisitos funcionales de la funcionalidad de consulta de estadísticas*

Vemos que se cumplen todos los requisitos funcionales definidos, lo cual es la medida de que la nueva funcionalidad se comporta de acuerdo a lo esperado.

En la tabla siguiente (tabla 8) verificamos el cumplimiento de los requisitos no funcionales definidos en el apartado 3.4.2.

Requisito	¿Se cumple?	Observaciones
RNF1	Sí	Se ha solicitado a varios usuarios familiarizados con el sistema la realización de un proceso de consulta de estadísticas y en ningún caso se han sobrepasado los 2 minutos de duración.
RNF2	Sí	Solamente se han definido dos nuevos estilos en la hoja de estilo *gades.css* para la tabla de la consulta de estadísticas: *Tabla* y *Celda*.
RNF3	Sí	Los usuarios que realizaron las evaluaciones del RF1 pudieron leer con éxito los textos y los rótulos y etiquetas de las gráficas a unos 60 cm de la pantalla.
RNF4	Sí	En los tres navegadores en los que se han realizado pruebas se han necesitado menos de 5 clics para imprimir la gráfica o tabla.
RNF5	Sí	En las tablas que contienen datos de evaluaciones no se muestran las iniciales del paciente, como podemos ver en la figura 81.
RNF6	Sí	Como se explica en el apartado 3.4.5, la solución respeta la arquitectura del sistema y se emplean las mismas tecnologías que se habían empleado antes del comienzo de este PFG.
RNF7	Sí	El API empleado para la generación de estadísticas, JFreeChart, es software libre y gratuito.
RNF8	Sí	El Javadoc de la nueva funcionalidad de consulta de estadísticas se puede encontrar en el CD-ROM adjunto a este libro.
RNF9	Sí	Se han realizado varios procesos de consulta de estadísticas con éxito en los navegadores Internet Explorer, Mozilla Firefox y Google Chrome.

Tabla 8. *Verificación de los requisitos no funcionales de la funcionalidad de consulta de estadísticas*

Todos los requisitos no funcionales que habíamos definido se cumplen, con lo cual los servicios o funciones del sistema cumplen las restricciones impuestas en la fase de diseño.

Dado que la funcionalidad desarrollada cumple todos los requisitos funcionales y no funcionales definidos en la fase de diseño, podemos considerar que la funcionalidad de consulta de estadísticas ha sido implementada con éxito.

5. Conclusiones

Este Proyecto Fin de Grado ha supuesto la continuación del proceso de desarrollo de las plataformas web de e-Salud Pegaso y Gades. Estas plataformas constituyen dos Sistemas Expertos cuya finalidad es tratar de facilitar la detección precoz de trastornos del lenguaje en niños de 0 a 6 años. Dichos sistemas están implementados siguiendo una arquitectura de tres capas, lo que hace que tengan un alto grado de modularidad y un bajo acoplamiento. Además, el uso de componentes Java EE en ambos sistemas favorece la reutilización de código.

El primero de los problemas planteados en este PFG ha sido la resolución de la incompatibilidad de Pegaso y Gades con Java SE 7. Ninguno de los sistemas podía realizar evaluaciones del lenguaje usando versiones de Java SE posteriores a Java SE 6. La incompatibilidad se ha detectado debido al reemplazo del algoritmo de ordenación *MergeSort* por *TimSort*, más eficiente, al pasar de Java SE 6 a Java SE 7. Este cambio ha dejado al descubierto la existencia de un *bug* en una de las librerías de Apache Jena, usadas para acceder a la ontología. La solución adoptada ha consistido en añadir un *script* en el servidor web Apache Tomcat que modifica la variable de entorno *CATALINA_OPTS*, indicándole a la JVM que emplee el algoritmo *MergeSort* en lugar del algoritmo *TimSort*. Esta solución permite que tanto Pegaso como Gades funcionen correctamente con Java SE 7.

La necesidad de independizar el proceso de generación de ontologías de los desarrolladores y encargados del mantenimiento de los sistemas Pegaso y Gades, nos lleva a la siguiente fase de este proyecto. Ésta ha consistido en el desarrollo de una aplicación capaz de generar una ontología en lenguaje OWL desde código Java. Dicha tarea, realizada de manera conjunta con Miguel Menéndez [MEN14], ha requerido la realización de una investigación del concepto de ontología y de los componentes de una ontología en lenguaje OWL, así como de los diferentes API que permiten generar ontologías OWL DL desde código Java. Nos hemos decantado por el uso de Apache Jena, por ser una solución sencilla y conocida, ya que estaba en uso en ambos sistemas desde antes del comienzo de este trabajo. Empleando los API proporcionados por Jena, se ha desarrollado una aplicación que genera una ontología reducida para los hitos de desarrollo del mes 1. La ontología generada se ha probado en el sistema Pegaso y ha demostrado ser válida para realizar inferencias en el mes 1. Esta aplicación ha sentado la base para la creación de una aplicación capaz de generar la ontología completa desde código Java, tarea que se ha continuado y documentado en [MEN14].

La tercera fase de este proyecto ha consistido en optimizar algunas de las funcionalidades ya desarrolladas para las plataformas Pegaso y Gades. Entre otras tareas, se ha añadido una opción que permite imprimir un informe con la información mostrada en el proceso la consulta de resultados.

Por último, como culminación de este proyecto, se ha llevado a cabo el análisis, diseño e implementación de una nueva funcionalidad para la explotación de los datos almacenados por los sistemas Pegaso y Gades. Esta nueva funcionalidad explota estadísticamente los datos correspondientes a las evaluaciones del lenguaje almacenadas en las BD de ambos sistemas. Así, a partir de dichos datos, la nueva funcionalidad elabora una serie de estadísticas que muestra al usuario en forma de gráficas y tablas.

La funcionalidad de consulta de estadísticas se ha desarrollado respetando la arquitectura original del sistema, por lo que los componentes y tecnologías utilizados han sido los mismos que en el resto de funcionalidades de Pegaso y Gades. Como explicábamos anteriormente, una de las bondades del enfoque en componentes es la reutilización de código, por lo que ha sido posible reutilizar gran parte del código que ya estaba presente en ambos sistemas. Este aspecto resulta, sin duda, muy beneficioso para los desarrolladores de Pegaso y Gades.

Después de llevar a cabo las pruebas oportunas, se ha comprobado que la solución desarrollada cumple con los requisitos funcionales y no funcionales definidos en la fase de diseño. Luego, se considera que la solución cumple los objetivos propuestos.

El proceso de mejora y ampliación de Pegaso y Gades llevado a cabo por este PFG concluye aquí, aportando fundamentalmente una aplicación que permite automatizar la generación de ontologías, y la capacidad de explotar los datos usados por los sistemas Pegaso y Gades en forma de estadísticas. Sin embargo, dado que el desarrollo de ambos sistemas no está cerrado, en el capítulo siguiente se proponen varias líneas de trabajo futuras.

6. Trabajo futuro

Con objeto de seguir evolucionando y mejorando los sistemas Pegaso y Gades, se proponen una serie de mejoras. Estas mejoras se explican en detalle a continuación:

- A pesar de que a día de hoy los SE Pegaso y Gades carecen de la capacidad de adquirir nuevo conocimiento por sí solos, el siguiente paso podría ser la obtención de conocimiento a partir de los datos almacenados en la BD.

Figura 85. Pirámide del conocimiento

Por tanto, se propone habilitar los mecanismos para que los SE sean capaces de analizar los datos almacenados en la BD para adquirir conocimiento a partir de éstos y poder modificar la BC por sí solos. La tarea de conseguir conocimiento a partir de los datos almacenados en la BD se conoce como Proceso de Descubrimiento de Conocimiento en Bases de Datos, **KDD** (*Knowledge Discovery in Databases*).

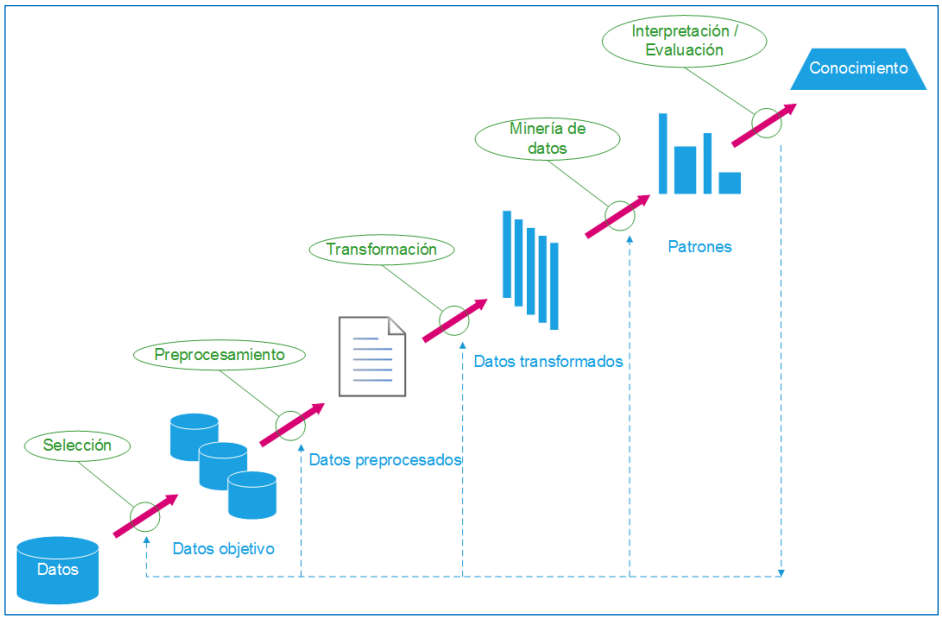

Figura 86. Proceso de Descubrimiento de Conocimiento en Bases de Datos (KDD) (elaboración propia a partir de [FAY97])

Para dicha tarea se puede considerar el uso de herramientas software como Weka, desarrollado en la Universidad de Waikato (Nueva Zelanda), y que constituye una colección de algoritmos de aprendizaje automático para realizar tareas de minería de datos [WEK14].

- Se propone reestructurar el módulo de consulta de estadísticas actual para otorgar al usuario la libertad de generar cualquier estadística. Es decir, se propone que el usuario pueda seleccionar un "campo X" (para el eje X) y un "campo Y" (para el eje Y) de las tablas "evaluacion" o "validacion" de la BD, de modo que pueda generar tantas estadísticas como combinaciones posibles de "campo X" y "campo Y" existan. En cualquier caso, podría ser razonable añadir alguna restricción, pues no todas las estadísticas posibles serían de interés.
- Se propone elaborar y poder consultar estadísticas relativas a las respuestas dadas, en las diferentes evaluaciones realizadas, a las preguntas correspondientes a los hitos recogidos en la BC. Por ejemplo, "porcentaje de respuestas afirmativas, negativas y ns/nc en las preguntas correspondientes a los hitos del mes 60".
- Se propone explotar la información relativa a todas las evaluaciones de un paciente y las valoraciones de las mismas, creando una especie de "historial clínico" del paciente en el sistema que pueda ser consultado por cualquier usuario.
- Se propone habilitar una opción para que los usuarios de los sistemas Gades y Pegaso puedan reportar errores detectados en ambos sistemas, realizar sugerencias y proponer cambios. Por ejemplo, se propone habilitar una casilla acompañando a las gráficas o tablas mostradas en el proceso de consulta de estadísticas para que los usuarios puedan marcar la estadística como útil o no. Opcionalmente, se propone añadir un campo a través del cual puedan realizar algún comentario que se almacene en la BD y pueda ser contrastado posteriormente con los comentarios de otros usuarios de cara a mantener o eliminar una estadística de la lista de estadísticas seleccionables.
- El aumento del número de accesos simultáneos a la BD, debido a la creación de nuevas funcionalidades para los sistemas Pegaso y Gades, puede llegar a imposibilitar los accesos a la BD y, por ende, imposibilitar la operación normal de los sistemas. En ese caso, probablemente se muestre por pantalla el mensaje de error "Too many connections".

  El número de conexiones simultáneas que se permiten realizar a la BD viene fijado, por defecto, en la variable *max_connections* de MySQL. El valor de esta variable depende de la versión del servidor MySQL usado; por ejemplo, en la versión del servidor MySQL 5.6, es 150. Aunque dicho valor se puede modificar, el problema

radica en que las conexiones que se realizan a la BD quedan en estado *sleep*, de modo que no son eliminadas tras haberse devuelto el resultado de la sentencia SQL. Se pueden eliminar manualmente usando el mandato *KILL CONNECTION [id_conexión]*, pero resulta una labor extremadamente tediosa e ineficiente. Por tanto, se propone implementar un mecanismo que permita eliminar conexiones a la BD en estado *sleep* automáticamente (por ejemplo, la creación de un *script* que se ejecute cada cierto tiempo de manera automática).

A la par que se implementa dicha solución, se propone estudiar la capacidad del servidor en el que se alojan los sistemas Pegaso y Gades para determinar el número máximo de conexiones simultáneas que puede soportar, actualizando en consecuencia la variable *max_connections* del servidor de BD MySQL y/o tomando las medidas oportunas para evitar una súbita caída de los sistemas.

7. Referencias

[ALO04] Alonso Betanzos, A.; *et al.* "La Ingeniería del Conocimiento", en "Ingeniería del Conocimiento: Aspectos Metodológicos". 1ª edición. Madrid: Pearson Educación, 2004. Cap 1, sec. 1.2, p. 11.

[BEN13] Benítez, R.; *et al.* "Introducción a la inteligencia artificial (IA)", en "Inteligencia artificial avanzada".1ª edición. Barcelona: Universitat Oberta de Catalunya, 2013. Cap. 1, sec. 1.1, pp. 10-11.

[CEW02] Cewolf [en línea]. *SourceForge.* Disponible en: <http://cewolf.sourceforge.net/new/index.html>

[FAY97] Fayyad, U.; *et al.* "From Data Mining to Knowledge Discovery in Databases". *AI Magazine.* 1996, vol. 17, núm. 3, p. 41. Disponible en: <http://www.aaai.org/ojs/index.php/aimagazine/article/view/1230/1131> doi: http://dx.doi.org/10.1609/aimag.v17i3.1230.

[GCH13] Google Charts [en línea]. *Google Inc.* Disponible en: <https://google-developers.appspot.com/chart/interactive/docs/index>

[GUI94] Guida, C.; Tasso, C. "Design and Development of Knowledge-Based Systems. From Life Cycle to Methodology", 1ª ed. Chichester: John Wiley and Sons Ltd., 1994.

[IBM12] Autor desconocido. "Ontologías, OWL, taxonomías y clasificaciones" [en línea]. IBM. 2012. Disponible en: <http://pic.dhe.ibm.com/infocenter/sr/v7r5/index.jsp?topic=%2Fcom.ibm.sr.doc%2Frwsr_configrn_classifications03.html> [Consulta: 12 de junio de 2014]

[JEN10] Jendrock, E.; et al. "The Java EE 5 Tutorial" [en línea]. Oracle. 2010. Disponible en: <http://docs.oracle.com/javaee/5/tutorial/doc/>

[JEN14] Apache Jena [en línea]. Apache Software *Foundation.* Disponible en: <https://jena.apache.org/>

[JFR05] Gilbert, D.; *et al.* JFreeChart [en línea]. *Object Refinery Ltd.* Disponible en: <http://www.jfree.org/jfreechart/>

[JFR07] Gilbert, D. "The JFreeChart Class Library: Developer Guide". *Object Refinery Ltd.* 2007.

[MAR11] Martín Ruiz, M. L. "Modelo de conocimiento para detección precoz de trastornos del lenguaje en niños de 0 a 6 años", Trabajo de Investigación. ETSIST-UPM, Madrid, 2011.

[MAR13] Martín Ruiz, M.L.; *et al.* "Arquitectura Telemática para la Detección Precoz de Trastornos del Lenguaje", XI Jornadas de Ingeniería Telemática, Granada, España, Octubre 2013.

[MAR14] Martín Ruiz, M.L.; *et al.* "Evaluating a Web-Based Clinical Decision Support System for Language Disorders Screening in a Nursery School", en Journal of Medical Internet Research, 2014. doi: 10.2196/jmir.3263.

[MEN14] Menéndez Álvarez, M. "Plataforma de conocimiento evolutivo para detección temprana de trastornos del lenguaje", Proyecto Fin de Grado. Dpto. Ing. y Arq. Telemáticas, ETSIST-UPM, Madrid, 2014.

[NEC91] Neches, R.; *et al.* "Enabling Technology For Knowledge Sharing". *AI Magazine*. 1991, vol. 12, núm. 3. Disponible en: <http://www.aaai.org/ojs/index.php/aimagazine/article/view/902/820> doi: http://dx.doi.org/10.1609/aimag.v12i3.902.

[ORA13] Autor desconocido. "Replace 'modified mergesort' in java.util.Arrays.sort with timsort" [en línea]. *Oracle*. 2009. Disponible en: <http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6804124>

[ORA14] Autor desconocido. "Java SE 7 and JDK 7 Compatibility" [en línea]. *Oracle*. Fecha desconocida. Disponible en: <http://www.oracle.com/technetwork/java/javase/compatibility-417013.html#behavioral>

[OWL14] OWL API [en línea]. *SourceForge*. Disponible en: <http://owlapi.sourceforge.net/>

[PRO10] Knublauch, H.; *et al.* Protege-OWL API Programmer's Guide [en línea]. *Protégé*. Universidad de Stanford. Palo Alto (Estados Unidos). 2010. Disponible en: <http://protegewiki.stanford.edu/wiki/ProtegeOWL_API_Programmers_Guide>

[PUE09] Puebla Hernández, I. "Arquitectura y diseño de un sistema de gestión de valores de bolsa", Trabajo Fin de Carrera. ETSIINF-UPM, Madrid, 2009. Disponible en: <http://oa.upm.es/1697/1/PFC_IRENE_PUEBLA_HERNANDEZ.pdf>

[RGR14] Heyes, R. RGrah: HTML5 charts library [en línea]. Disponible en: <http://www.rgraph.net/>

[SAM04] Samper Márquez, J.J. "Introducción a los Sistemas Expertos". *REDcientífica* [en línea]. Disponible en: <http://www.redcientifica.com/imprimir/doc199908210001.html>. ISSN: 1579-0223

[STA14] Stallman, R. "Por qué el «código abierto» pierde de vista lo esencial del software libre" [en línea]. *GNU & FSF*. Disponible en: <https://www.gnu.org/philosophy/open-source-misses-the-point.html#TransNote1> [consulta: 15 de junio de 2012]

[STR12] Autor desconocido. "Representing Information Using the Web Ontology Language" [en línea]. 2012. Disponible en: <https://content.scottstreit.com/Semantic_Web/Slides/0510_OWL%20-%20byLacy%20-%20Section%203a.ppt>

[TOM7] Autor desconocido. "Running The Apache Tomcat 7.0 Servlet/JSP Container" [en línea]. *Apache Software Foundation*. Fecha desconocida. Disponible en: <http://tomcat.apache.org/tomcat-7.0-doc/RUNNING.txt>

[UC311] Villena Román, J.; *et al.* "Tema 3: Sistemas Basados en Conocimiento", [en línea]. Universidad Carlos III de Madrid, Madrid. Disponible en: <http://ocw.uc3m.es/ingenieria-telematica/inteligencia-en-redes-de-comunicaciones/material-de-clase-1/03-sistemas-basados-en-conocimiento>

[URI13] Martín Uría, J. "Implementación de una arquitectura de componentes para un sistema de apoyo a la toma de decisiones en Atención Primaria", Proyecto Fin de Grado. Dpto. Ing. y Arq. Telemáticas, ETSIST-UPM, Madrid, 2013.

[VAD07] Vadillo Moreno, L. "Creación y simulación de un Módulo de Razonamiento para un Sistema de Teleasistencia en el Hogar Digital", Proyecto Fin de Carrera. Dpto. Ing. y Arq. Telemáticas, ETSIST-UPM, Madrid, 2007.

[VIZ10] Vizcarra Romero, J.C. "Repositorio semántico de datos espaciales", Tesis de Máster. Centro de Investigación en Computación, IPN, Ciudad de México, 2010. Disponible en: <http://www.saber.cic.ipn.mx/cake/SABERsvn/trunk/Repositorios/webVerArchivo/595/1>

[WEL09] Autor desconocido. "CATALINA_OPTS v JAVA_OPTS - What is the difference?" [en línea]. *Well House consultants Ltd.* 2009. Disponible en: <http://www.wellho.net/mouth/2163_CATALINA-OPTS-v-JAVA-OPTS-What-is-the-difference-.html>

[WEK14] Weka 3: Data Mining Software in Java [en línea]. Universidad de Waikato, Hamilton (Nueva Zelanda). Disponible en: <http://www.cs.waikato.ac.nz/ml/weka/>

[WIK1] Colaboradores de Wikipedia. "Aplicación informática" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Aplicación_informática> [Consulta: 20 de mayo de 2014]

[WIK2] Colaboradores de Wikipedia. "Cliente-servidor" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Cliente-servidor> [Consulta: 20 de mayo de 2014]

[WIK3] Colaboradores de Wikipedia. "Modularidad" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Modularidad> [Consulta: 24 de julio de 2014]

[WIK4] Colaboradores de Wikipedia. "Acoplamiento" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Diseño_estructurado#Acoplamiento> [Consulta: 27 de mayo de 2014]

[WIK5] Colaboradores de Wikipedia. *Software de código abierto* [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Software_de_código_abierto> [Consulta: 27 de mayo de 2014]

[WIK6] Colaboradores de Wikipedia. "Java Runtime Environment" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Java_Runtime_Environment> [Consulta: 29 de mayo de 2014]

[WIK7] Colaboradores de Wikipedia. "Depuración de programas" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Depuración_de_programas> [Consulta: 30 de mayo de 2014]

[WIK8] Colaboradores de Wikipedia. "Script" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Script> [Consulta: 31 de mayo de 2014]

[WIK9] Colaboradores de Wikipedia. "Serialización" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Serialización> [Consulta: 2 de junio de 2014]

[WIK10] Colaboradores de Wikipedia. "Pseudocódigo" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Pseudocódigo> [Consulta: 4 de junio de 2014]

[WIK11] Colaboradores de Wikipedia. "Base de datos" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Base_de_datos> [Consulta: 6 de junio de 2014]

[WIK12] Colaboradores de Wikipedia. "Información" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Información> [Consulta: 6 de junio de 2014]

[WIK13] Colaboradores de Wikipedia. "Javadoc" [en línea]. *Wikipedia.* Disponible en: <http://es.wikipedia.org/wiki/Javadoc> [Consulta: 7 de julio de 2014]

[W3C04] McGuinness, D.L.; van Harmelen, F. Web Ontology Language Overview [en línea]. W3C. 2004. Disponible en: <http://www.w3.org/TR/2004/REC-owl-features-20040210/>

Anexo I: Aplicación Java para generar una ontología OWL

GeneradorOntologias.java

A continuación se presenta el código de la aplicación Java que genera una ontología OWL reducida para Pegaso (ver apartado 3.2).

```
import java.io.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.vocabulary.OWL;

/**
* Esta aplicación 'stand alone' es capaz de generar una ontología en
* lenguaje OWL DL válida para las preguntas del mes 1 (año 1)
* de las aplicaciones Gades y Pegaso, obra de Mª Luisa Martín Ruiz como
* parte de su Tesis Doctoral.
*
* @author Oscar Palomo Díaz & Miguel Menéndez Álvarez
* ETSIS de Telecomunicación - Universidad Politécnica de Madrid
* Fecha: marzo de 2014
* Version: 1.1
*/
public class GeneradorOntologias{
      //Espacio de nombres (URI) que identifica a esta ontología
      private static String ns = "http://ontologiaMes1.owl#";

      /**
       * Método principal del generador de ontologías.
       * @param args
       */
      public static void main(String[] args){
            //Creamos el modelo de ontología OWL DL
            OntModel m =
            ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM);
            //Especificamos el sufijo de la etiqueta "xmlns"
            m.setNsPrefix("", ns);
            /*Añadimos la cabecera de la ontología
            (<owl:Ontology rdf:about=""/>)*/
            m.createOntology("");

            //Creamos la clase "AvanceSL" y sus subclases
            OntClass avanceSL = m.createClass(ns + "AvanceSL");
            OntClass primerAño = m.createClass(ns + "PrimerAño");
            OntClass meses1 = m.createClass(ns + "Meses1");
            OntClass meses1al1 = m.createClass(ns +
                                    "AL_NoReaccionaAUnaCampana");
            OntClass meses1al2 = m.createClass(ns +
                                    "AL_NoVocalizaSinLlorar");
            avanceSL.addSubClass(primerAño);
            primerAño.addSubClass(meses1);
            meses1.addSubClass(meses1al1);
            meses1.addSubClass(meses1al2);
```

```
        //Creamos la clase "DecisionSistema" y sus subclases
        OntClass decisionSistema = m.createClass(ns +
        "DecisionSistema");
        OntClass hito = m.createClass(ns + "Hito");
        OntClass hitoAlarma = m.createClass(ns + "HitoAlarma");
        OntClass año1 = m.createClass(ns + "Año1");
        OntClass decision = m.createClass(ns +
        "EnviarEspecialistaParaComprobarAudicion");
        decisionSistema.setSuperClass(OWL.Thing);
        decisionSistema.addSubClass(hito);
        hito.addSubClass(hitoAlarma);
        hitoAlarma.addSubClass(año1);
        año1.addSubClass(decision);

        //Creamos los individuos
        Individual alarma1mes1 = meses1al1.createIndividual(ns +
        "AL_I_NoReaccionaAUnaCampana");
        meses1al2.createIndividual(ns + "AL_I_NoVocalizaSinLlorar");
        Individual decisionSistemaInd =
        decisionSistema.createIndividual(ns + "DecisionSistema_1");

        //Creamos la propiedad "hayRespuestaNegativaEn"
        ObjectProperty propiedad = m.createObjectProperty(ns +
        "hayRespuestaNegativaEn");
        propiedad.addDomain(decisionSistema);
        propiedad.addRange(avanceSL);
        decisionSistemaInd.setPropertyValue(propiedad, alarma1mes1);

        //Creamos las restricciones de propiedad
        //Creamos la restricción 'allValuesFrom'
        AllValuesFromRestriction all =
        m.createAllValuesFromRestriction(null, propiedad, avanceSL);
        decisionSistema.addSuperClass(all);
        //Creamos la restriccion 'someValuesFrom'
        /*Creamos una clase compleja con las dos preguntas usando el
        constructor 'unionOf'*/
        RDFNode[] preguntas = {meses1al1, meses1al2};
        RDFList lista = m.createList(preguntas);
        UnionClass union = m.createUnionClass(null, lista);
        SomeValuesFromRestriction some =
        m.createSomeValuesFromRestriction(null, propiedad, union);
        decision.setEquivalentClass(some);

        //Escribimos la ontología en un archivo de extensión .owl
        escribir(m);
    }
```

```
	/**
	 * Método encargado de escribir en el formato deseado la ontología
	 * creada.
	 * @param m Ontología creada, preparada para ser guardada en un
	 * fichero.
	 */
	public static void escribir(OntModel m){
		FileWriter out = null;
		try{
			//Elegimos el nombre y la ubicación del archivo .owl
			out = new FileWriter("H:/tmp/ontologiaMes1.owl");
			/*"RDF/XML-ABBREV" produce una salida legible sin prestar
			mucha atención a la eficiencia*/
			RDFWriter writer = m.getWriter("RDF/XML-ABBREV");
			/*Tipo de URIs relativos usados en el documento: "same-
			document"*/
			writer.setProperty("relativeURIs", "same-document");
			/*El valor del espacio de nombres 'xml:base' es el
			contenido de la variable 'ns'*/
			writer.setProperty("xmlbase", ns);
			//Escribimos el archivo .owl
			writer.write(m, out, null);
		}
		catch (IOException e){
			e.printStackTrace();
		}
		finally{
			if (out != null){
				try{
					out.close();
				}
				catch (IOException ignore){}
			}
		}
	}
}
```

ontologiaMes1.owl

A continuación se presenta la ontología OWL reducida generada para Pegaso usando la aplicación *GeneradorOntologias.java*.

```
<?xml version="1.0" encoding="windows-1252"?>
<rdf:RDF
    xmlns="http://ontologiaMes1.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://ontologiaMes1.owl#">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Hito">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="DecisionSistema"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AL_NoVocalizaSinLlorar">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Meses1"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AL_NoReaccionaAUnaCampana">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Meses1"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Año1">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="HitoAlarma"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#HitoAlarma">
    <rdfs:subClassOf rdf:resource="#Hito"/>
  </owl:Class>
  <owl:Class rdf:ID="PrimerAño">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="AvanceSL"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#DecisionSistema">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#AvanceSL"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hayRespuestaNegativaEn"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:about="#Meses1">
    <rdfs:subClassOf rdf:resource="#PrimerAño"/>
  </owl:Class>
```

```
  <owl:Class rdf:ID="EnviarEspecialistaParaComprobarAudicion">
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class>
            <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="#AL_NoReaccionaAUnaCampana"/>
              <owl:Class rdf:about="#AL_NoVocalizaSinLlorar"/>
            </owl:unionOf>
          </owl:Class>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hayRespuestaNegativaEn"/>
        </owl:onProperty>
      </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Año1"/>
  </owl:Class>
  <owl:ObjectProperty rdf:about="#hayRespuestaNegativaEn">
    <rdfs:range rdf:resource="#AvanceSL"/>
    <rdfs:domain rdf:resource="#DecisionSistema"/>
  </owl:ObjectProperty>
  <AL_NoVocalizaSinLlorar rdf:ID="AL_I_NoVocalizaSinLlorar"/>
  <DecisionSistema rdf:ID="DecisionSistema_1">
    <hayRespuestaNegativaEn>
      <AL_NoReaccionaAUnaCampana rdf:ID="AL_I_NoReaccionaAUnaCampana"/>
    </hayRespuestaNegativaEn>
  </DecisionSistema>
</rdf:RDF>
```

Anexo II: Manual de usuario de la funcionalidad de consulta de estadísticas en Gades

Herramienta de ayuda a la decisión para facilitar a las Escuelas Infantiles la detección precoz de niños con trastornos del lenguaje

Manual de usuario de la funcionalidad de consulta de estadísticas

El presente manual de usuario describe la funcionalidad de consulta de estadísticas del sistema Gades. Ha sido realizado utilizando el navegador web Google Chrome. Tenga en cuenta que el aspecto de algunas páginas puede variar ligeramente usando otros navegadores web.

Acceso al sistema

La página de acceso al sistema Gades, accesible en la URL http://marisa.diatel.upm.es/gades/, se muestra en la figura 87. Para acceder, es necesario estar dado de alta en el sistema como logopeda o educador infantil.

Introduzca su nombre de usuario (en "Usuario") y su contraseña (en "Contraseña"), y a continuación pulse "Enviar".

Figura 87. *Página de acceso al sistema Gades*

En caso de que el usuario introducido no exista, aparecerá el siguiente mensaje de error en pantalla:

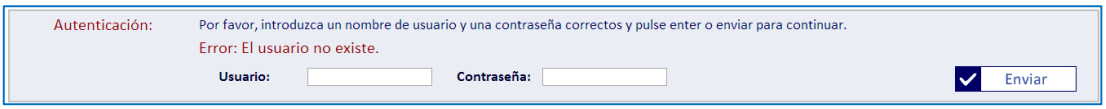

Figura 88. *Mensaje de error mostrado tras introducir un usuario que no existe*

En caso de que la contraseña introducida no sea correcta, se mostrará el siguiente mensaje de error en pantalla:

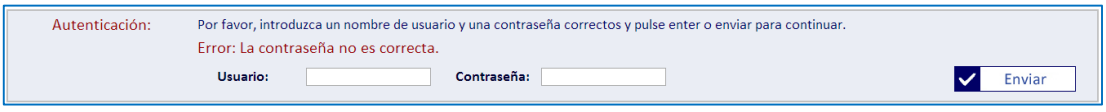

Figura 89. *Mensaje de error mostrado tras introducir una contraseña incorrecta*

Consulta de estadísticas

Una vez autenticado, ya puede seleccionar cualquiera de las funcionalidades del sistema. Seleccione "CONSULTAR ESTADÍSTICAS" (aparece recuadrada en rojo en la figura 90).

Figura 90. Página de bienvenida de Gades

A continuación, seleccione el centro educativo y el profesional a los que pertenecen las estadísticas de las evaluaciones del lenguaje que desea consultar. Puede elegir tres tipos de combinaciones:

- **Estadísticas relativas a un único profesional de un único centro educativo**. En la figura 91 se muestra un ejemplo.

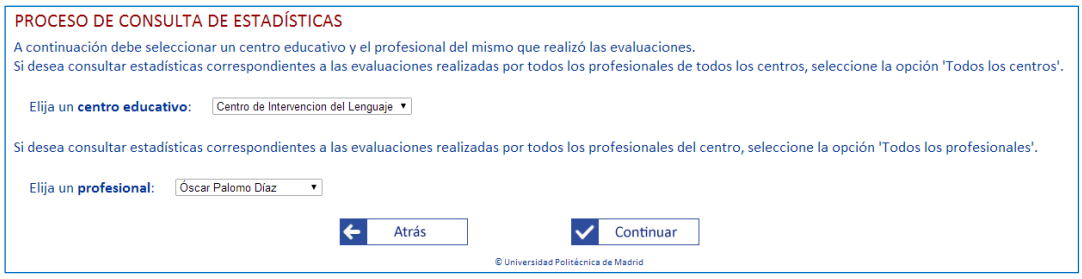

Figura 91. Selección de centro y profesional en la de consulta de estadísticas

- **Estadísticas relativas a todos los profesionales de un único centro educativo**. Elija un centro educativo en la lista desplegable de centros educativos y a continuación seleccione la opción "Todos los profesionales" en la lista desplegable de profesionales, como se muestra en la figura 92.

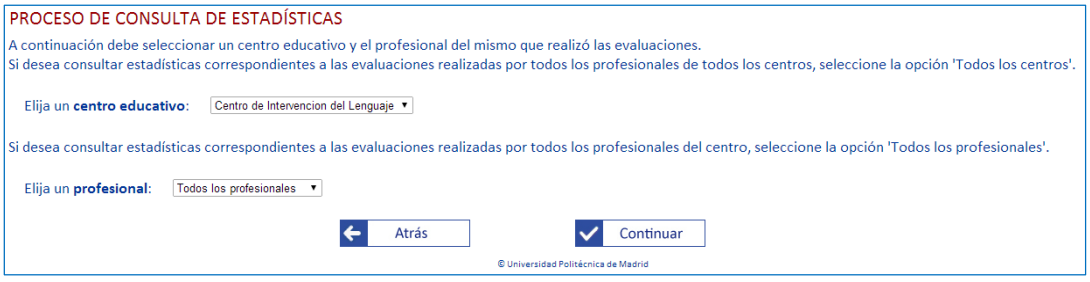

Figura 92. Selección de todos los profesionales de un centro en la consulta de estadísticas

- **Estadísticas relativas a todos los profesionales de todos los centros educativos.** En este caso, seleccione la opción "Todos los centros" en la lista desplegable de centros educativos, como se muestra en la figura 93. La selección de esta opción implica la elección de todos los profesionales, por lo que no se muestra la lista desplegable de profesionales.

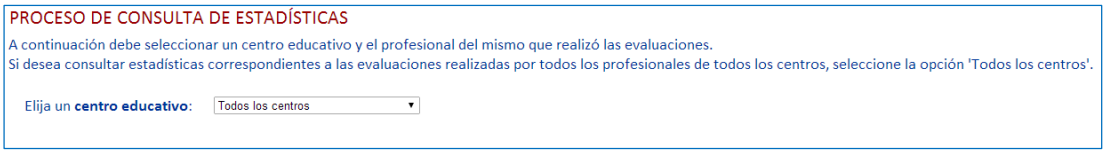

Figura 93. Selección de todos los profesionales de todos los centros en la consulta de estadísticas

En caso de que no haya seleccionado ni centro educativo ni profesional, se mostrará el siguiente mensaje de error en pantalla:

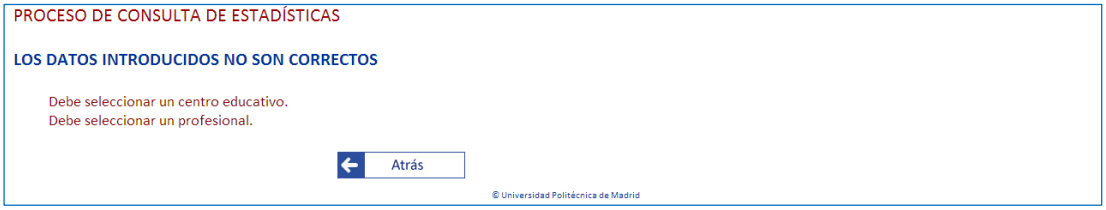

Figura 94. Mensaje de error mostrado tras no seleccionar ni centro ni profesional

Tras pulsar el botón "Continuar", puede seleccionar la estadística que desee consultar.

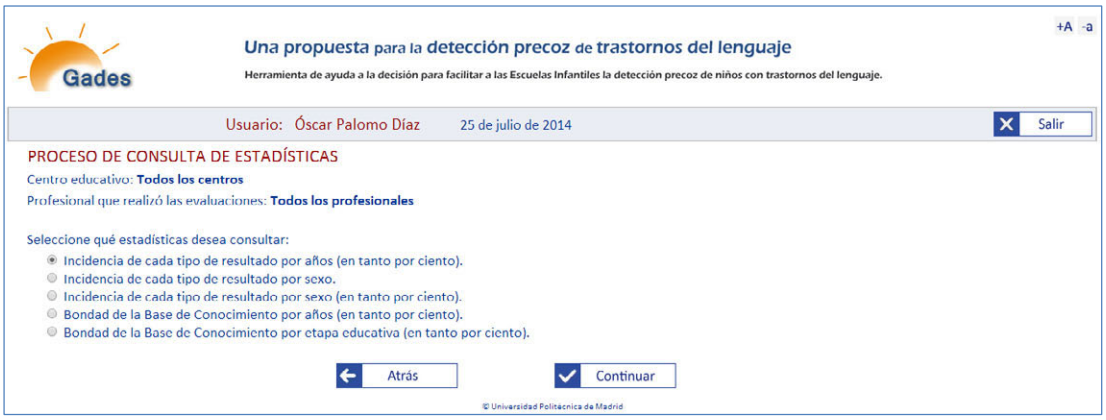

Figura 95. Página de selección de la estadística que se desea consultar

Tenga en cuenta que la última opción ("Tabla resumen de las evaluaciones realizadas por el profesional seleccionado") solo se muestra si ha seleccionado un único profesional de un único centro educativo. Si ha seleccionado todos los profesionales de un único centro educativo o de todos los centros educativos, esta opción no estará disponible.

Una vez seleccionada la estadística deseada, pulse el botón "Continuar" para ver en pantalla la información relativa a la estadística seleccionada. En la figura 96 se muestra un ejemplo.

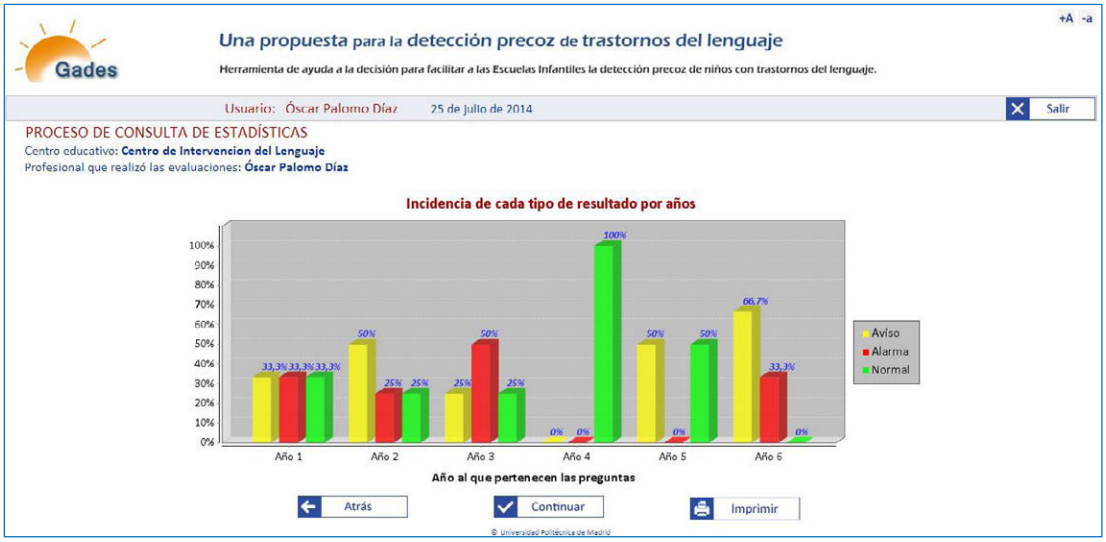

Figura 96. Pantalla en la que se muestra la estadística elegida

Si desea volver a la pantalla inicial, puede pulsar el botón "Continuar".

En caso de haber seleccionado un profesional que aún no ha realizado ninguna evaluación del lenguaje, se mostrará el siguiente mensaje en pantalla.

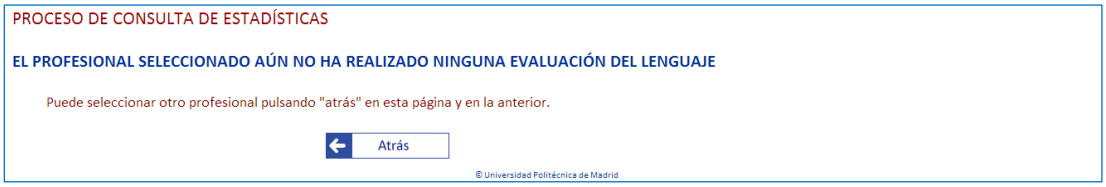

Figura 97. Mensaje de error mostrado tras seleccionar un profesional que aún no ha realizado ninguna evaluación del lenguaje

Para seleccionar otro profesional, pulse el botón "Atrás" en esta página y también en la anterior.

Impresión de la estadística consultada

Si desea imprimir la gráfica o tabla mostrada en pantalla, pulse el botón "Imprimir" (ver figura 98). Tras pulsarlo, aparecerá un cuadro de diálogo en el que podrá seleccionar la impresora de destino o si desea guardarlo como PDF (tenga en cuenta que este cuadro puede variar en función del navegador web empleado).

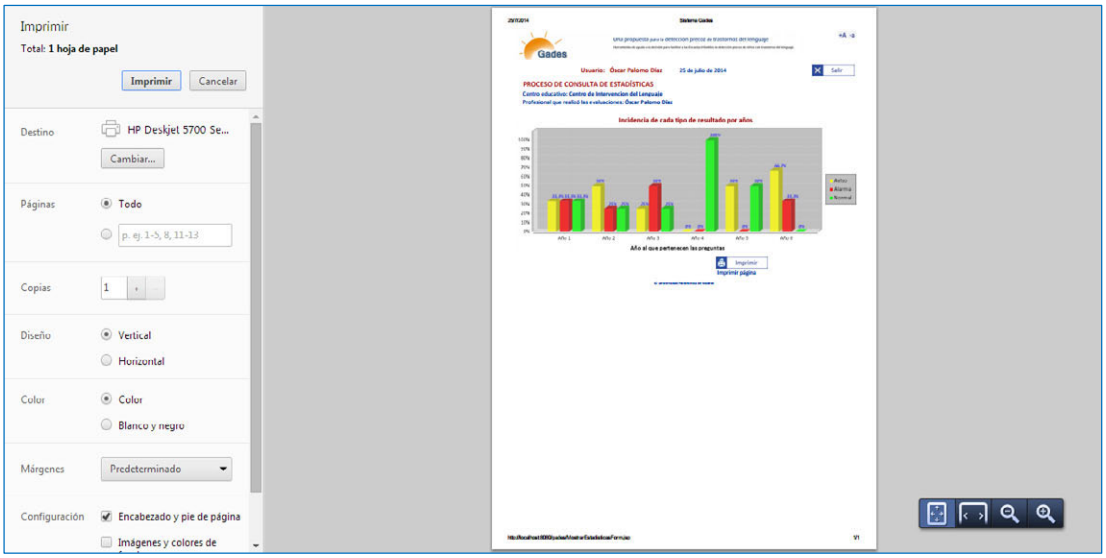

Figura 98. Pantalla de impresión de la estadística elegida

Volver a la página anterior

En cualquiera de las páginas (salvo en la inicial), puede volver a la página anterior pulsando el botón "Atrás" ubicado en la parte inferior de la pantalla.

Figura 99. Botón para volver a la página anterior

Salir del sistema

Para abandonar la aplicación en cualquier momento durante el proceso de consulta de estadísticas, pulse el botón "Salir" ubicado en la parte superior de la pantalla.

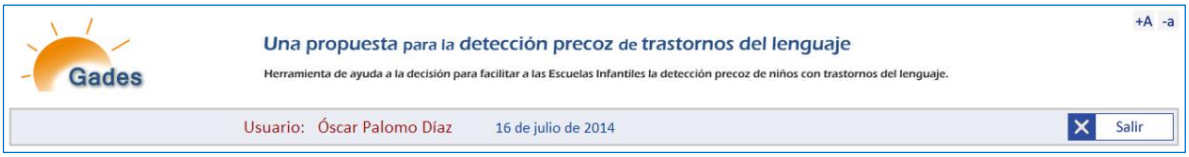

Figura 100. Botón para salir del sistema

No obstante, antes de abandonar el sistema le aparecerá un cuadro de diálogo en el que podrá confirmar si desea salir (pulsando "Aceptar") o si, por el contrario, desea abortar la operación (pulsando "Cancelar").

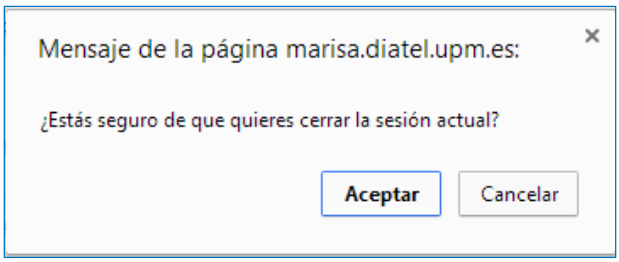

Figura 101. *Cuadro de diálogo para confirmar si desea salir del sistema*

Al salir de la aplicación, volverá a la página inicial de la misma. Si, por el contrario, cancela la operación, se quedará en la página en la que estaba.

En la página 73, la tabla 5, en la columna "Descripción" de la fila correspondiente al atributo "valoraciones", **dice**:

"Valoraciones de las evaluaciones del lenguaje realizadas"

Debe decir:

"Valoraciones de las decisiones del sistema correspondientes a las evaluaciones del lenguaje realizadas"